MacTech Magazine
Vol. 19, No. 04 • April 2003

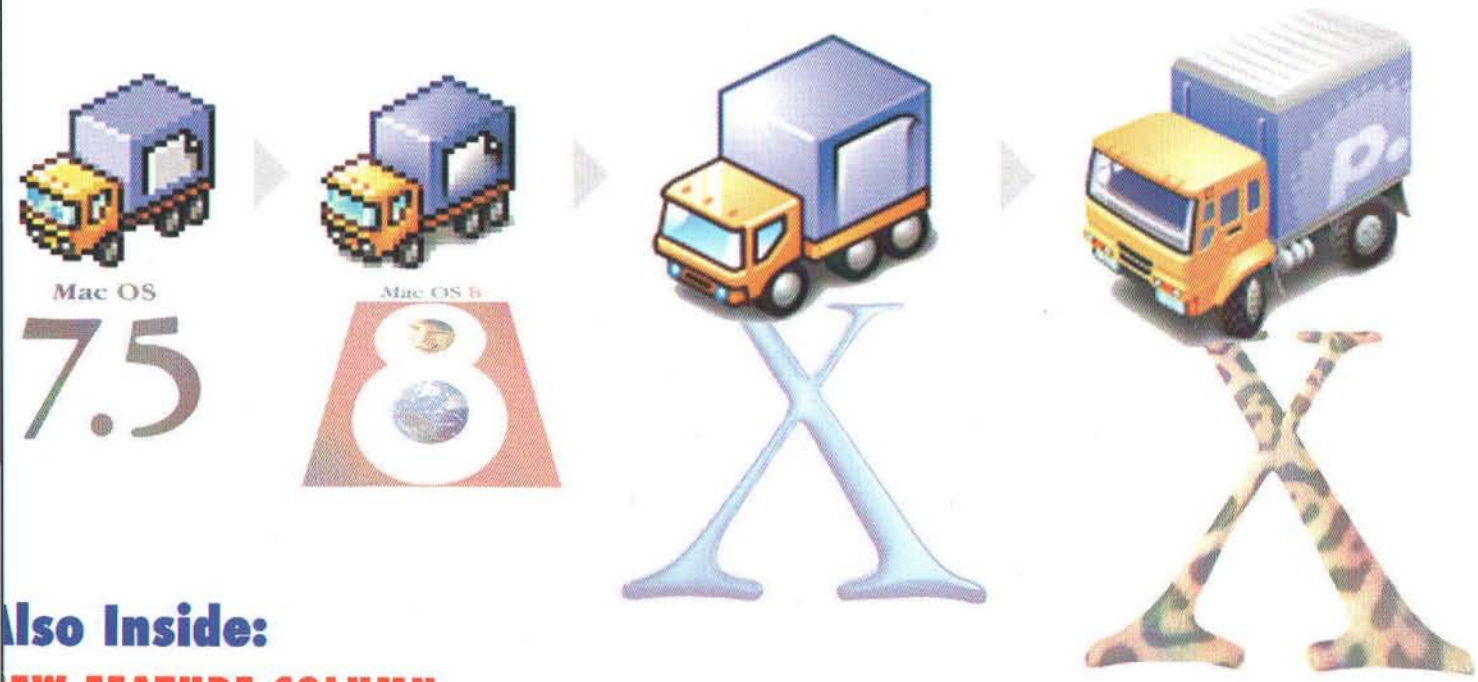
MacTech®

The Journal of Macintosh Technology and Development

Redesigning Right

Adapting an Interface for Mac OS X

By Cabel Sasser



Also Inside:

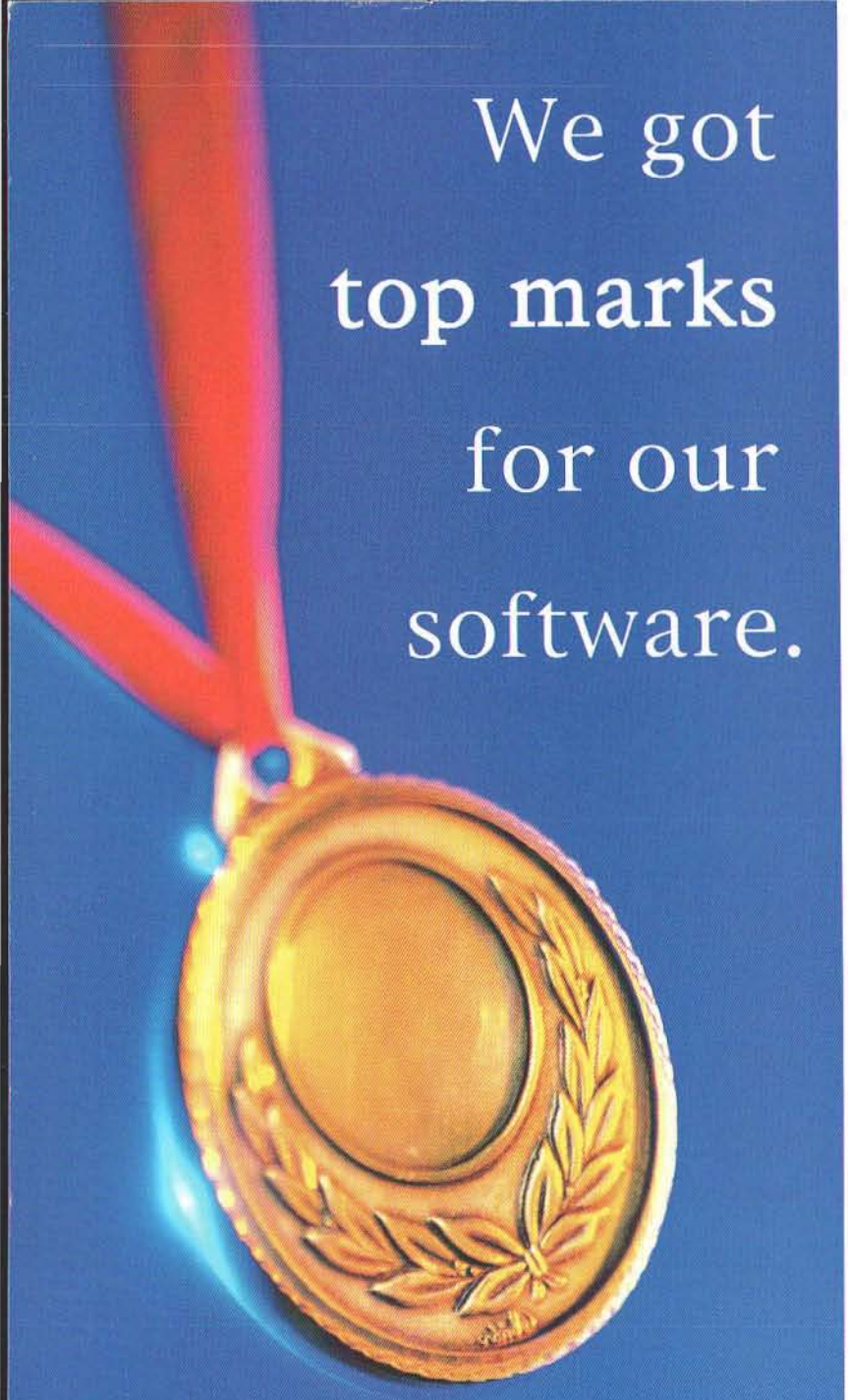
NEW FEATURE COLUMN

Mac OS X Programming Secrets
by Scott Knaster

Books on Cocoa Programming
by Rich Morin

Jaguar for Administrators
by John C. Welch

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



We got top marks for our software.

You can too.

Revolution™
The Solution for Software Development
In enterprise, business and education

REVOLUTION™

Limitless possibilities

With Revolution, the solutions you can create are endless.

True cross-platform development

With the click of a button, you can build applications for Macintosh (Classic and OS X), Windows, Linux, and Unix.

Increased productivity

The powerful interface builder and easy-to-use programming language speed up all the essentials of software development, leaving you with more time to focus on your project.

Databases, multimedia, Internet applications, external libraries and more

Revolution supports everything you need: SQL databases, streaming media, control of QuickTime, QTVR and graphics, CGI processing, Internet protocols, and more.

FREE Trial Version

www.runrev.com

MacUser UK - 5 mice

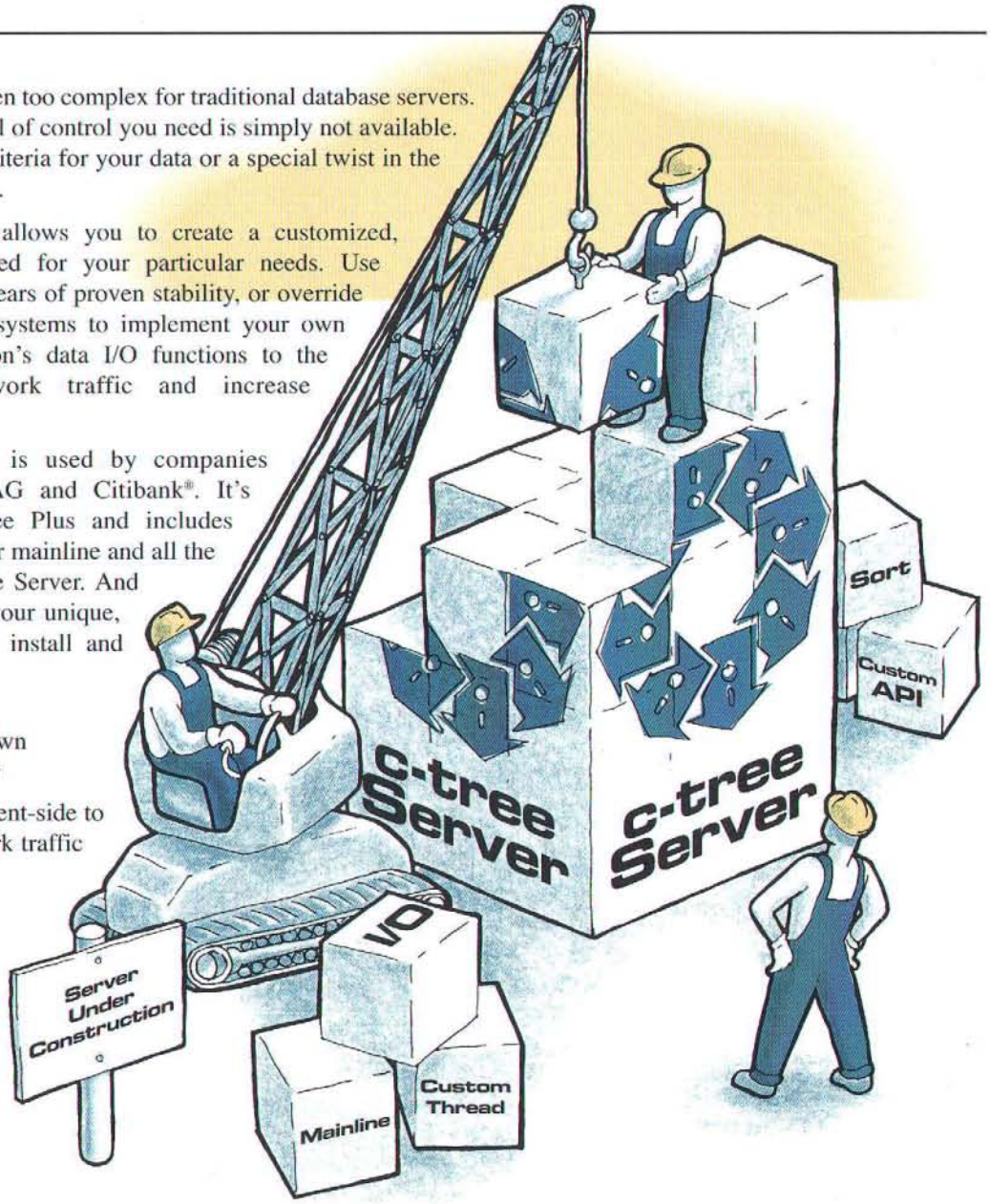
RUNTIME REVOLUTION

CUSTOMIZE
YOUR DATABASE SERVER WITH THE C-TREE® SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree® Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank®. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!



- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing

Visit www.faircom.com/ep/mt/sdk today to take control of your server!

FairCom®
www.faircom.com

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • info@faircom.com

HELPING FILEMAKER PRO CUSTOMERS SCALE TO NEW HEIGHTS.

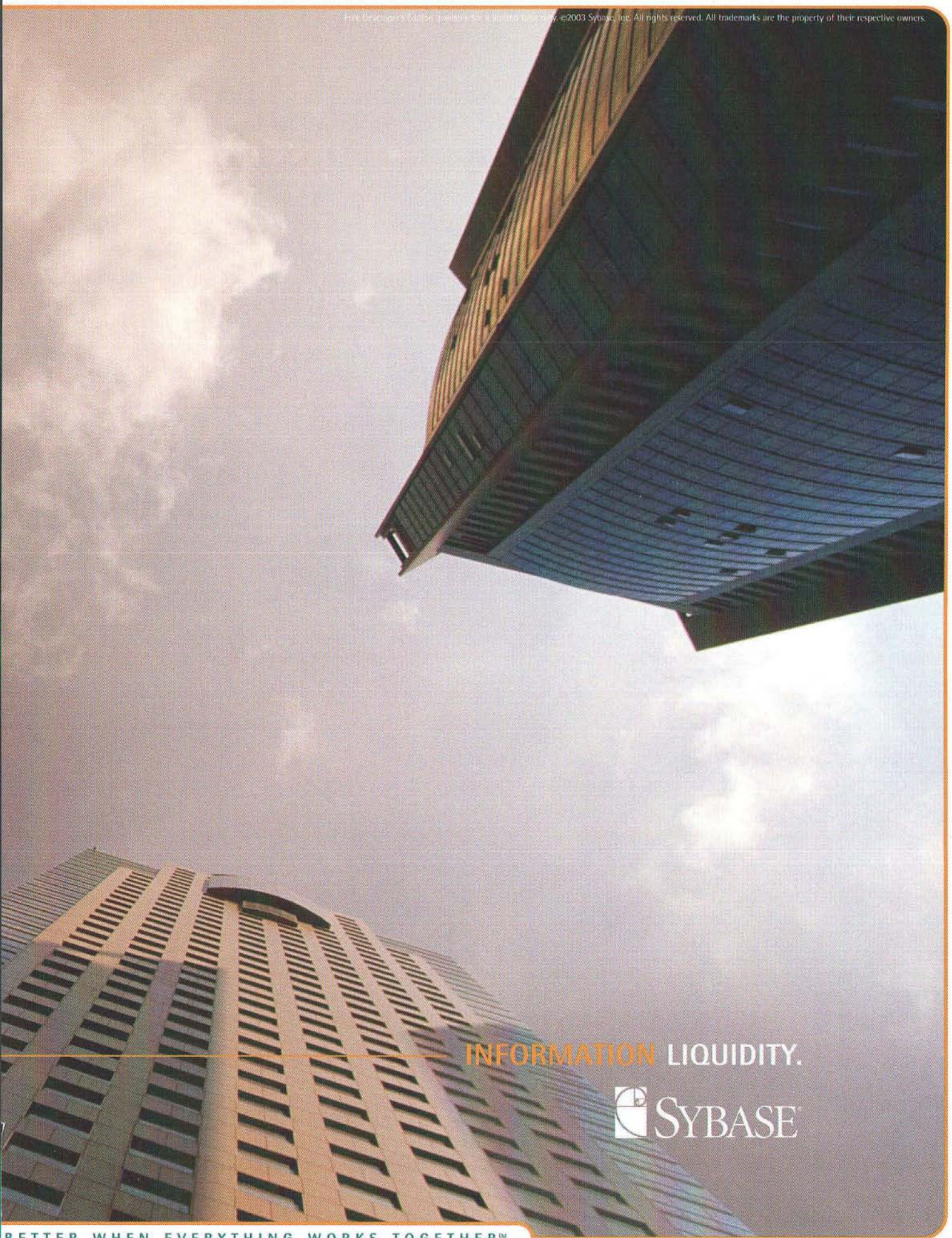
Want to enhance the performance of FileMaker Pro on MAC OS X? The same database engine that runs Wall Street can help you do just that.

ASE 12.5 increases the reliability, availability and scalability of your FileMaker Pro application. It provides better data integrity, world-class security and the ability to handle thousands of transactions per minute. It'll also give your users the power of SQL queries.

ASE 12.5 for MAC OS X is yet one more example of how Sybase is helping today's leading companies achieve Information Liquidity: a highly profitable state where all of your information is transformed into real economic value.

A FREE Developer's Edition download is available online at sybase.com/filemaker.

SYBASE INTEGRATION TECHNOLOGIES. EVERYTHING WORK



MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

April 2003 • Volume 19, Issue 4

Getting Started: Project Builder Revealed

In this month's column, we're going to explore some of Project Builder's nooks and crannies. The column started on a completely different tack, when I found myself trying to find the definition for a specific class function. I dug through the /Developer/Documentation/ directory, which is absolutely worth doing, but there's a *lot* of doc there.

I went on line to ask some of my Project Builder buddies how they would go about solving this particular problem and got a bunch of different responses, all of which led to the same end, and each of which taught me something new about Project Builder. This month, I'd like to share what I've learned with you.

Creating CrannyTester

Let's start off by creating a new project, called CrannyTester. Launch Project Builder, select New Project... from the File menu, scroll down to the bottom of the New Project window and create a Foundation Tool project. Name the project CrannyTester.

One Window, Some, or Many?

One Project Builder feature I really like is the ability to customize PB's use of windows. Select Preferences... from the Project Builder menu, then click on the Task Templates icon at the top of the prefs window. Now click on the Basic Settings tab (**Figure 1**).

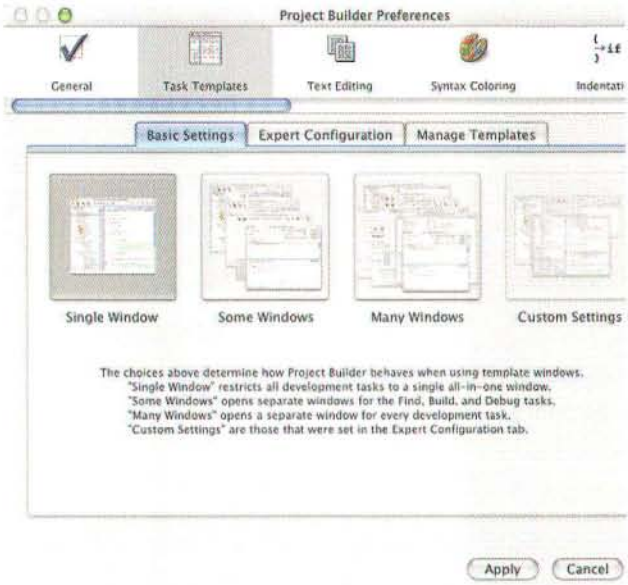


Figure 1. *The window template settings in the Preferences... dialog.*

As you can see, there are four basic window setups. Single Window forces the entire Project Builder interface into a single swiss-army-knife of an interface, a single window filled with tabbed panes (**Figure 2**). If you are working on a laptop or on a smaller monitor, this is a very efficient way to go.

Dave Mark is a long-time Mac developer and author and has written a number of books on Macintosh development, including ***Learn C on the Macintosh***, ***Learn C++ on the Macintosh***, and ***The Macintosh Programming Primer*** series. Be sure to check out Dave's web site at http://www.spiderworks.com.

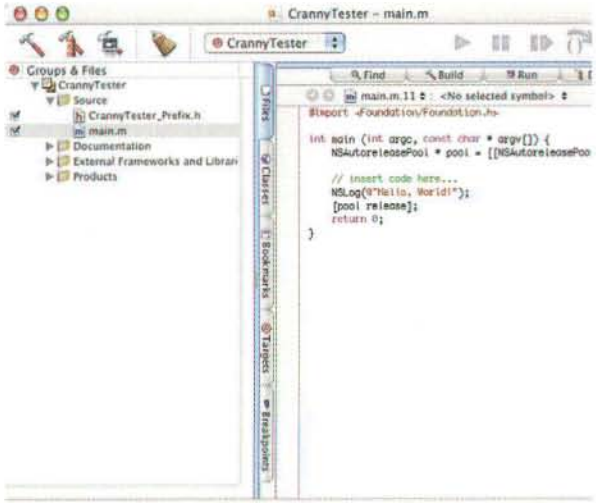


Figure 2. *The CrannyTester project in Single Window mode.*

One bit of funkiness to be aware of: When you change selections in the Task Templates preference panes, your changes won't take effect until you open a new window. To get the change to affect your current project, make the change, then close your project and reopen it.

To see this for yourself, set the preference to Single Window, then open a project or create a new one. A set of horizontal tabs (similar to the Find/Build/Run/Debug/CVS tags in **Figure 2**) will appear in the window.

Now go into preferences and change to Some Windows. Even if you click the Apply button, the tabs will still be visible. Now close the project and reopen it. The tabs disappear as the Some Windows preference is applied.

Chances are, you'll pick a mode you like, then create all your projects in that mode. If you do find yourself playing with the windowing modes, be sure to reopen the project to see the effect properly.

"Some Windows" adds separate windows for Find, Build and Debug. "Many Windows" opens everything in its own window and feels most like CodeWarrior to me. I spend most of my time in Some Windows but love the flexibility of switching to Single Window when I'm on the road. Sweet!

Finding the Doc

Now that you've got your project setup the way you like it, I'd like to share another Project Builder nook with you. Or maybe it's a cranny. Hmmm.

Open the CrannyTester project you created at the beginning of this month's column. Click on the Files tab, open the Source triangle, and click on the source file main.m. Here's the source code you should see:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
  NSAutoreleasePool *pool=[[NSAutoreleasePool alloc] init];
```

```
    // insert code here...
    NSLog(@"Hello, World!");
    [pool release];
    return 0;
}
```

This is the default source for a Foundation Tool and should look reasonably familiar to you. One of the keys to understanding any new framework is the ability to find descriptions of the classes and utility functions that you encounter in the framework's documentation. One thing Apple's dev tools have no shortage of is documentation. While I *do* recommend that you spend some time prowling through the /Developer/Documentation/ subdirectories, there are some clever little shortcuts built into Project Builder that will bring you right to the pages you are looking for.

Let's start with the most obvious path. If you click on the Classes tab, you'll see a list of classes in the upper pane, with a list (empty, right now) of members in the lower pane. Below that is a popup menu that should be set to "Heirarchy, all classes" with an Options button to its right (See **Figure 3**.)

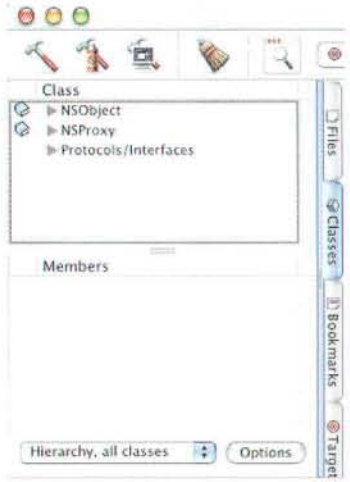


Figure 3. *The Classes tab, with its list of classes.*

Lets say you wanted to read up on the NSObject class (and you should). NSObject is the root of most of the classes you'll use to develop your Cocoa applications. Click on NSObject in the Class pane. You'll immediately see a list of its member functions in the lower pane. Click on a member function, and that function's declaration appears in the main editing pane.

Now for the coolness. Notice that little blue book icon to the left of the NSObject entry in the Class pane? Click on it. Ka-ching! The NSObject documentation appears in the editing pane. Apple's documentation is extensive. For each class, you'll learn where the class fits into the class hierarchy, protocols it conforms to, in what include files it is declared, methods and fields (with lots of "see also" links), and more. At the very least, spend some time reading about the NSObject class, since the vast majority of your classes will inherit from this class.

Here's another example: Go back to the Class pane and click on the triangle to the left of NSObject. This will reveal the classes derived from NSObject. Near the top of that list is NSArray. Click on the triangle to the left of NSArray, revealing NSMutableArray. Click on the book icon for NSMutableArray. The doc for NSMutableArray will appear in the main pane (see **Figure 4**).

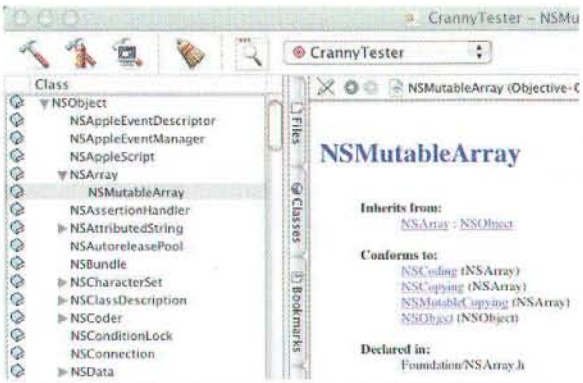


Figure 4. *The documentation for NSMutableArray.*

Notice that the doc lists the inheritance hierarchy NSArray : NSObject. As you can see, you've got several ways to find your class in the hierarchy. You can use the triangles to drill down through a parent class. You can also use the "Inherits from" links in the doc to get to an ancestor class.

Now take a look at the popup menu at the bottom of the Members pane. It currently reads "Hierarchy, all classes". There are some other choices as well. Selecting "Hierarchy, project classes" will restrict the class list to classes you've declared in your project. "Flat, all classes" gives you a straight alphabetical listing of all classes, irrespective of hierarchy. This is very useful when you know the name of a class, but don't know where it sits in the class tree. "Flat, project classes" does the same thing, but restricts the list to classes used by your project.

Another nice Project Builder feature is that you can add your own customized views to this list. **Figure 5** shows the dialog that appears when you click the Options button to the right of the popup menu. Basically, you use the controls in the dialog to customize the class listing, then click the Add... button. When prompted, name your custom listing, and it will appear in the popup menu along with the other listings. Spend a few minutes playing with this dialog, just so you have a sense of what options are available to you.

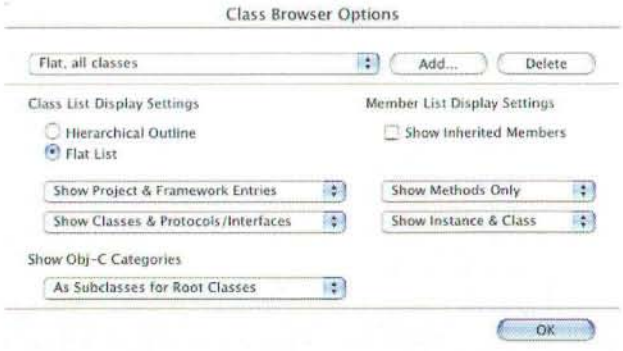


Figure 5. *This dialog appears when you click the Options button in the Classes tab.*

Other Paths to the Documentation

As you might expect, there are other ways to get from a symbol to its documentation. In your CrannyTester project, click on the Files tab, then open the Source triangle and click on main.m. Now hold down the option key and double-click on the class name NSAutoreleasePool.

Ta-daa! Project Builder takes you straight to the doc page for NSAutoreleasePool. To get back to your source code, click on the "back" button (the button the cursor is pointing to in **Figure 6**). Go ahead, click on it. Just like your browser, it takes you back to the previous page, in this case, your source code.

Figure 6. *The cursor is pointing to the back button.*

Let's try another experiment. This time, hold down the command key and double-click on NSAutoreleasePool. Instead of jumping to the doc, Project Builder jumps to the declaration of NSAutoreleasePool in NSAutoreleasePool.h.

Notice the two popup menus to the right of the back and forward browser buttons. The first popup (**Figure 7**) lists the recently visited frames, including source code and documentation. The second popup (**Figure 8**) allows you to jump to individual declarations within the current source file.

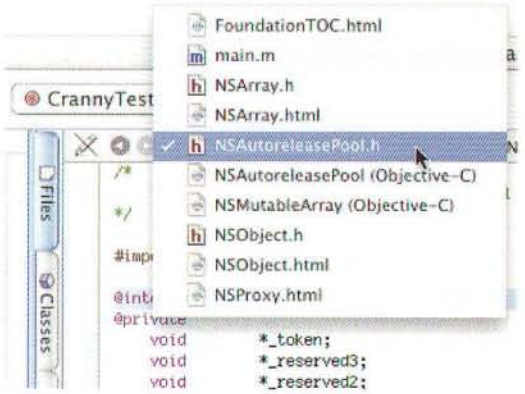


Figure 7. *The first popup to the right of the browser buttons shows recently visited files.*

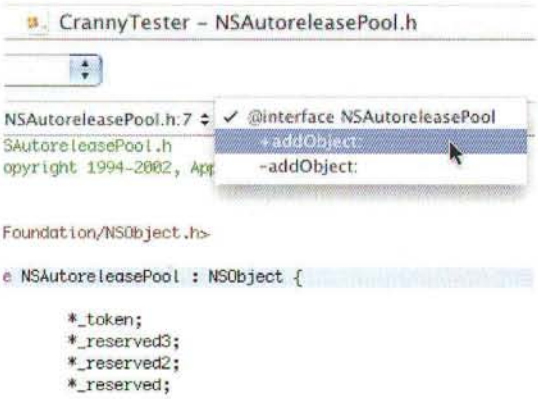


Figure 8. *The second popup lets you jump to the declarations within the current file.*

Till Next Month...

There are a lot more elements worth exploring within Project Builder but, unfortunately, I've run out of room. More cool stuff next month, plus we'll dig into some actual code. See you then. ☺

By Rich Morin

Tk: A Portable GUI Toolkit

expressiveness and simplicity count for a lot

Last month, I mentioned Tk, referring to it as an X11 toolkit. This month, I'd like to cover Tk in a bit more detail. I should begin, however, by clarifying that Tk is not just an X11 toolkit; in fact, it is a platform- and language-independent GUI toolkit.

Tk can be used with C, Perl, Python, Ruby, Tcl (Tk's original counterpart), and many other programming languages. It can be used with Aqua, PalmOS, X11, and assorted Microsoft APIs. In summary, any modern computer with the appropriate hardware probably has Tk support.

Tk also wins big on expressiveness and simplicity. Most GUI toolkits (e.g., Cocoa and assorted X11 "widget sets") are complex and have steep learning curves. Tk, in contrast, is remarkably simple to program, once a few basic concepts have been mastered. Although Tk may not have all the features of Cocoa, it supports most of the widgets you are likely to need; it also has ways to add new widgets, if you're feeling sufficiently motivated.

Tcl/Tk

Because Tk started as a toolkit for the Tcl language, most of the Tk resources you'll find are actually for Tcl/Tk, the language/toolkit combination. Also, because Tcl/Tk is the focus of development, the other language bindings (e.g., Perl/Tk, Tkinter) occasionally lag behind. Finally, if you want to extend Tk, you'll probably find yourself coding some Tcl in the process.

In short, you should plan to learn a bit of Tcl, even if you intend to use Tk with some other language. Fortunately, this isn't particularly painful; Tcl syntax only takes about five minutes to learn (most of which you'll spend accepting the idea of how simple the syntax really is :-).

Pick up a copy of the Mac OS X Tcl/Tk port from:

http://www.apple.com/downloads/macosx/unix_open_source/tcltk.html

Be sure to look at the ReadMe files; they contain a number of useful pointers and critical details. After the (dmg, mpkg) installation finishes, you should find a copy of "Wish Shell":

/Applications/Utilities/Wish Shell.app

When you start up the app, two windows will appear. The "Console" window allows you to run Tcl commands, viewing the output:

```
set a 123
123
() 2 % set b [expr $a * 2]
246
() 3 %
```

Tcl isn't a conventional language and it isn't processed in a conventional manner. Although some parts of Tcl syntax look a bit like C or Perl, the interpreter is simply performing a sequence of commands and string substitutions. Thus, the first line sets a variable ("a") to the text string ("123"). You may think of this as a numeric value, but the Tcl interpreter doesn't.

The second line does three things. First, it performs a "variable substitution", replacing "$a" with the value "123". Next, it performs a "command substitution", replacing the expression "[expr $a * 2]" with the result (246). Finally, it sets "b" to the resulting value.

Now, let's bring Tk and the other (Wish Shell) window into play:

```
() 3 % button .b -text "Hello, world!" -command exit
.b
() 4 % pack .b
() 5 %
```

If all went as expected, the "Wish Shell" window should now contain a button labeled "Hello, world!". Pushing that button should cause the Wish Shell to exit (and both of its windows to disappear). In two lines of code, you have created and run a (tiny) windowing application!

If you simply want to program in Tcl/Tk, you are now in a position to do so. The results will look nicely Aquafied (Aquatic?) and you don't have to tell anyone how little code you wrote! If you become intrigued with Tcl/Tk, be sure to pick up John Ousterhout's excellent (if dated) "Tcl and the Tk Toolkit"

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.

(Addison-Wesley). John created both Tcl and Tk, so he's eminently qualified to say how they should be used.

Perl/Tk

Although Tcl is syntactically interesting, extremely portable, and quite flexible, you may prefer (as I do) to do your actual programming in some other language. Perl/Tk allows Perl programmers to use Tk widgets, while keeping the rest of their code in Perl. As hinted above, analogous libraries are available for C, Python, Ruby, and other programming languages.

Having said this, I should note that Tk calls may not be a smooth fit into the syntax of the chosen language. Here is a Perl/Tk translation of our simple window example:

```
use Tk;

my ($b, $mw);

$mw = new MainWindow;
$b  = $mw->Button(-text    => 'Hello, world!',
                  -command => sub { exit; });
$b->pack();
MainLoop();
```

This code is substantially larger, but the original Tk commands and arguments are still present (albeit buried in Object-Oriented Perl syntax :-). Several lines of overhead code bring in the Tk module, declare a couple of variables, create a window, and finally, start up the Tk event loop. Also note that we are no longer using the Tcl `exit` command; instead, we have created an anonymous Perl function which uses Perl's own `exit` command.

If you want to play with Perl/Tk, your first step should be to purchase a copy of "Learning Perl/Tk" (Nancy Walsh, O'Reilly). Buy Ousterhout's book, too, if you can; you won't regret it!

Now for the bad news: installing Perl/Tk on Mac OS X is quite complex and (worse) still in a state of flux. It is possible, with some difficulty, to make Perl/Tk work with Apple's X11 server and Developer Kit. I don't know of a way to get Perl/Tk to work with the native Tk port (as installed above), but I hope to have more information by the time this column is in print. Look for instructions on

http://www.cfcl.com/rdm/section7/perltk.html

I also suggest, for the brave-hearted, the following links:

https://lists.sourceforge.net/lists/listinfo/tcl-mac
http://tcl.activestate.com/ftp/mac
http://tcl.sourceforge.net (MacTclTk, TclTkAqua)
http://www.scriptics.com
http://www.tcl.tk/software/mac/AppleScript.html
http://www.tcl.tk/software/mac/index.html

MAC OS X PROGRAMMING SECRETS

By Scott Knaster

I am honored, privileged, and more than a little thrilled to welcome Scott Knaster to the fold here at MacTech. In the Mac programming universe, Scott is a legend, both for his sense of humor and for books like Macintosh Programming Secrets and How to Write Macintosh Software. While I was busy writing about C and the basics of the Mac Toolbox, Scott was taking folks to the next level, writing about off screen drawing, spinning cursors, VBL tasks, dialog filters, etc.

And that's just what we're gonna do here. While *Getting Started* will focus on the basics, *Mac OS X Programming Secrets* will take you to the next level. And for those of you who have yet to experience Scott, trust me. You are in for a treat!

Dave Mark
Editor-in-Chief

Thanks, Dave, for that awesome introduction. I'll try mightily to live up to those high expectations. And now, let's get started. But how to begin?

[Option #1: the wacky two-liner.] Well, I haven't written anything about Mac programming since about 1990. So as I was saying...

[Option #2: the clueless.] Hi. I've been basically out of the Mac universe for the last 10 years. Did anything interesting happen? Is Apple still in business? Excuse me... the company is run by *who?* Mac OS is now based on *what?* Hahahahahahaha!!! You big kidder!

[Option #3: keeping it real.]

That's me: always with the joking around. But really, when I think about writing for MacTech, my whole MacLife (not the same as iLife) flashes before my eyes, and I ponder how I got started on the Mac, how I ended up back here again, and what happened in between.

Programming Is Great Fun

My story, dear reader, is one of extreme good fortune and a personality tic. The lucky part came back in 1984 when I managed to find a job in the developer support department of the Macintosh division at Apple. I knew a little about programming and was in love with working at Apple, but I had never seen anything like the Mac – of course, very few of us had.

On my first day, my boss, Cary Clark, showed me the professional Mac development environment of the era. It consisted of a Mac with 128K RAM, a Lisa connected to the Mac for compiling and building applications in Pascal, and an Apple III (or Apple ///, as we insisted on spelling it) used as a dumb terminal for talking to the object-code debugger. Yes, it took three-fourths of Apple's computer models just to say "Hello World" on an underpowered original Mac. I was in awe.

I spent the next 3 years helping developers get their apps running on those early Macs. Because I was lucky enough to be in the right place at the right time, I learned a ton of cool stuff about Mac programming and debugging, like how to figure out what was on the stack, what really happened when heaps were compacted, and what caused system errors. Because lots of people were interested in learning these things, I wrote it all down in a couple of books, thus freeing my brain cells for important work, such as knowing it was Randy Van Warmer who sang the smash hit "I'd Really Love To See You Tonight".

Attentive readers will recall that way back a few paragraphs ago, I said it was luck and a personality tic that helped my career. The luck was finding myself in Mac developer support; the tic is that when I learn about cool new technology, like Mac programming, I just gotta tell other people about it. I can't stand keeping it to myself. This is a wonderful trait for a tech support guy and a writer.

Eventually, I got distracted by a shiny object or something, and I wandered away from the Mac for awhile. I wrote many charming but now-useless technical docs, first about Pink, one of Apple's several failed fix-the-Mac-OS crusades, and then about Magic Cap, a really wonderful OS and UI toolkit for personal communicators that just didn't catch on. Most surprising of all journeys, I ended up spending a number of happy years at Microsoft (yes, *that* Microsoft), at first working on Mac software, then writing books about Windows user interface guidelines, of all things.

New Things Happened

And then, Mac OS X sprang forth. A hypnotically pretty user interface, a foundation based on Unix (go figure), and lots

---

Scott Knaster has been writing about Macs for as long as there have been Macs. Scott's books ***How To Write Macintosh Software*** and ***Macintosh Programming Secrets*** were required reading for Mac programmers for more than a decade. Scott wrote developer books for General Magic and worked on Mac software for Microsoft. Scott's books have been translated into Japanese and Pascal. Scott has every issue of Mad magazine, which explains a lot.

of attention and cool new apps came with it. Apple's dark days had come to an end. Although I was making my living on that other platform, I was always drawn back to the Mac. When Apple started shipping Macs with OS X pre-installed in 2001, I bought an iBook and had a torrid affair with the new operating system. Although it wasn't ready for prime time then, you could tell that it had a great future – kind of how it felt using a Mac in 1984 or so. For the next two years, I always found time to keep up with OS X, messing around with development tools and generally having fun with all the new stuff Apple kept tossing our way.

When I said farewell to my employment at Microsoft (and my WinXP lifestyle) earlier this year, I fell down the OS X rabbit hole at full velocity. I found myself getting up early and staying up late, playing with all the toys: Project Builder, Terminal, Cocoa, Carbon, etc. Mac programming today is a surreal fusion of traditional Apple stuff and full-on Unix, and we haven't even begun to tap all the power that's in there. My goal in writing this column is to stay one step ahead, to show you something each month that you didn't know about before.

I find myself cursing Unixy things like folder permissions and command line syntax booboos, even while I enjoy trying to figure them out. Placing Unix under Aqua is a little like having a hood on my car. Mostly I just drive the car, but once in a while I like to open the hood to learn about or tweak something inside. Sometimes that's frustrating, but it's always educational.

I'm past the point where operating systems are a religion with me. I lived happily on last millennium's Mac OS for a long time, Windows for a few years, and now it's Mac OS X. There are things I like and dislike about each of them. I enjoy having all of them around, because they're fun to delve into and they all help me get work done. Although it's far from perfect, OS X might be my favorite OS ever because there's just *so much stuff* to mess with. And because after all these years, I find that my brain is still tuned to Apple's frequency: I keep coming back to Apple products because they usually *feel right* to me.

There Is Plenty To Do

So the truth is this column is my excuse to keep diving into OS X programming and then inflicting the results upon you, dear reader, because I just can't keep it to myself. There's so much to write about: great tools, cool tricks with frameworks, little-known APIs, debugging tips, odd and fun nooks in the system. Everything is new again, and we're all bozos on this bus. I want to write about all of it at once, but unfortunately the laws of physics prohibit this, so I'll do it one column per issue.

Speaking selfishly, I know I'm going to have a great time, and I hope you find some of it useful or amusing, because that would please the publisher and editors of MacTech. So thanks for coming along.

And I now that I think of it, it was England Dan & John Ford Coley who sang the smash hit "I'd Really Love To See You Tonight". I guess I better start writing more stuff down to free up brain space.

By Neil Ticktin, Publisher

Serious Wireless

Looking to get Point-to-Point Wireless between locations? Here's a truly robust solution.

Why The Change?

Some of you may recall my first wireless article a few years back (*MacTech Magazine*, January 2000 issue, "Going Wireless!" <http://www.mactech.com/articles/mactech/Vol.16/16.01/GoingWireless/>). In that article, I wanted to implement a reliable wireless link between my house, and the office. I specifically wanted to do anything and everything that I could to avoid the phone company for my net connection. Wireless gave me the ability to get high quality, reliable access that was completely within my control, and without any monthly fees. Sounded good to me!

And it *was* great. For years, I reliably had a 1.4 mbps connection between my house and my office ... a drive of about 2.5 miles, and about 1.5 miles as the bird flies. I was able to do it with off the shelf materials ... Lucent boxes (very similar technology to what's in the original Airport Base Stations, but with Lucent software to glue it all together).

The problem is, 1.4 mbps is not what it was 3+ years ago. More importantly, even though I was physically fairly close to the office, I was suffering latencies in the 30-millisecond range. Now while that's not a problem for things like web browsing, certain functions like access to our FileMaker Server were way too slow for me to use the way I wanted to.

You see, some applications move a lot of very small pieces of data around. The theory we had with our FileMaker sluggishness is that the latency was too great (especially compared to the LAN at the office), and essentially, it was causing the network to "thrash" when certain functions performed. As we saw in installing the new setup, reducing latency made a huge difference in the way these systems worked.

By the way, in case you were thinking about utilizing the new 802.11g standard at 54 mbps, beware of something. 802.11g is great for indoor LANs, but the standard is still in flux, and more importantly, the amount of distance you can get (at least inside) is about 1/3 of what is available with 802.11b. So far, I've not heard of anyone using 802.11b for outdoor point-to-point access.

The New Plan

You may remember from the previous article that I don't have line of sight from my house to the office. The solution? I knocked on the door of a neighbor up on the hill and offered him a great Internet connection in exchange for having some equipment on his roof. As you might well imagine, that's a deal that works out well for all. So, unlike many installations, I actually have a point to point-to-point connection between my house and the office. In other words, my connection goes from my house to David's house, to the office ... point to point to point.

What I needed to do now is find new equipment that could reduce the latency, as well as increase the bandwidth. I turned to my friends over at Westlink Wireless http://www.westlinkwireless.com. Eddie West helped me to take a look at the problem and recommended that we look at some of the proprietary radios out there.

The Radios

We selected the Trango Broadband radios <http://www.trangobroadband.com>. Specifically, the Access5800 Wireless Broadband Access Solution. Specifically, we're using a combination of their "Access Points" (AP) and their "Subscriber Units" (SU). The APs are typically located at the center of a network, or head-end of a wireless point of presence, and they communicate with one or more SUs. You can have up to 500 SUs per AP unit. We're using the Access Point, M5800S-AP-60, $895; and a 3 Mile Subscriber Unit, M5800S-SU, $495. (There's also a 10-mile Subscriber Unit, which has a DirecTV style dish on it.)

Radios have a certain amount of range where the signal is "visible". In Trango's terms, their AP units have a 60-degree sector. Because my house and my office are not within the same sector of visibility (they are on opposite sides of David's house), I need to have to Access Points each talking to a Subscriber Unit (4 radios in all). Put another way, the AP at my house talks to a SU at David's which is connected through a small Ethernet switch to an AP at David's which talks to an SU at my office.

The radios, and antennae are all combined into a single, weatherproof unit. No power is necessary on the roof, only

Neil is the publisher of MacTech Magazine. As a closet geek, he tends to experiment with some of the more interesting forms of technology, and then frightening the magazine staff by announcing he'll write about them. You can reach him at publisher@mactech.com

at the power injector, which is likely near your router or switch. The units use a "power injector" over Ethernet. The radio can be 100 meters from the power injector. The injector has three connections — a power cable, and two Ethernet ports (one to your LAN and one to the radio). For some installations, not having power at the radio is critical ... and it's almost always handy.

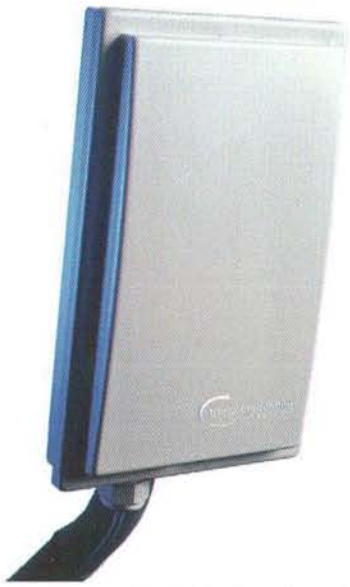


Figure 1. *Access Point (the 3 mile Subscriber Unit)*

Figure 2. *Antenna for 10 mile Subscriber Unit Configuration*

Radio Tech Stuff

These Trango radios are pretty cool. A specific benefit to us is that they are not based on one of the 802.11 standards, and are therefore they were able to do some special things, are high performance and secure. Specifically, they are based on Direct Sequence Spread Spectrum technology that operates in the ISM 5.7 - 5.8 GHz range. They theoretically deliver up to 10 Mbps user throughput reliably. An Access Point can be up to 20 miles from the SU, and supports QoS and have on the fly bandwidth throttling. Spread spectrum has the inherent advantage over other forms of transmission in that it can still recover data even in the most noisy of RF environments where other architectures fail. This isn't something new, just something very reliable and robust.

Using a proprietary protocol called SMARTPolling, the Trango radios are able to discover one another, as well deliver bandwidth more efficiently. And, if you are concerned with balancing the signal between SUs at different distances, the APs have a "power leveling" feature. As part of the "power on" sequence, the AP determines how far away a given SU is, and actively diminishes or increases its output power to ensure a quality of service.

To keep usage under control, you can optimize the Committed Information Rate / Maximum Information Rate (CIR/MIR) and provide bandwidth management/throttling per individual SU.

One of the more interesting features is the Dynamic RF Packet Sizing allows the radios to optimize data bandwidth utilization with maximum RF sensitivity. Unlike other solutions which use fixed RF packet sizes, the Access5800's protocol dynamically analyzes each Ethernet packet and transmits at an optimal length, 64 bytes (in the case of URL requests) to 1600 bytes.

SU's can be remotely managed through the AP using a host of commands or via HTTP web interface, telnet, or serial interface. The radios have a built in RSSI LED and RSSI Telnet Command for SU antenna alignment.

And, there's a fallback channel provision on the SU which offers built in redundancy. In the event an SU cannot communicate with its primary AP, the SU will intelligently search for a pre-programmed fall back AP to communicate with. This feature allows the network operator to actually sleep at night knowing that each user's network link will still be up in the morning.

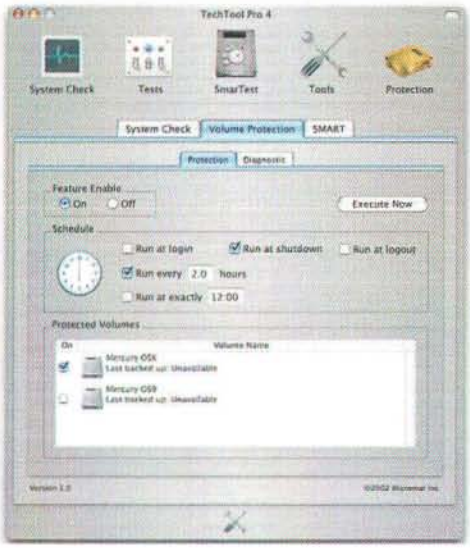
But there are two features that are super cool, especially if you are familiar with installing wireless networks. First, there are internal dual polarized antennas that are software switchable — yes, you can change the polarity through the software interface! Second, the radios have built in site survey tools that allow you to check for interference (see **Listing 1**).

Listing 1: Site Survey Results

```
ss> survey 3 v

Running site survey for 30 secs.
Press [space] then [enter] to stop
```

Get the MOST From Your Mac!



TECHTOOL PRO

New Version 4!
The next generation of the best-selling problem-solving utility for Macintosh.

MICROMAT INCORPORATED

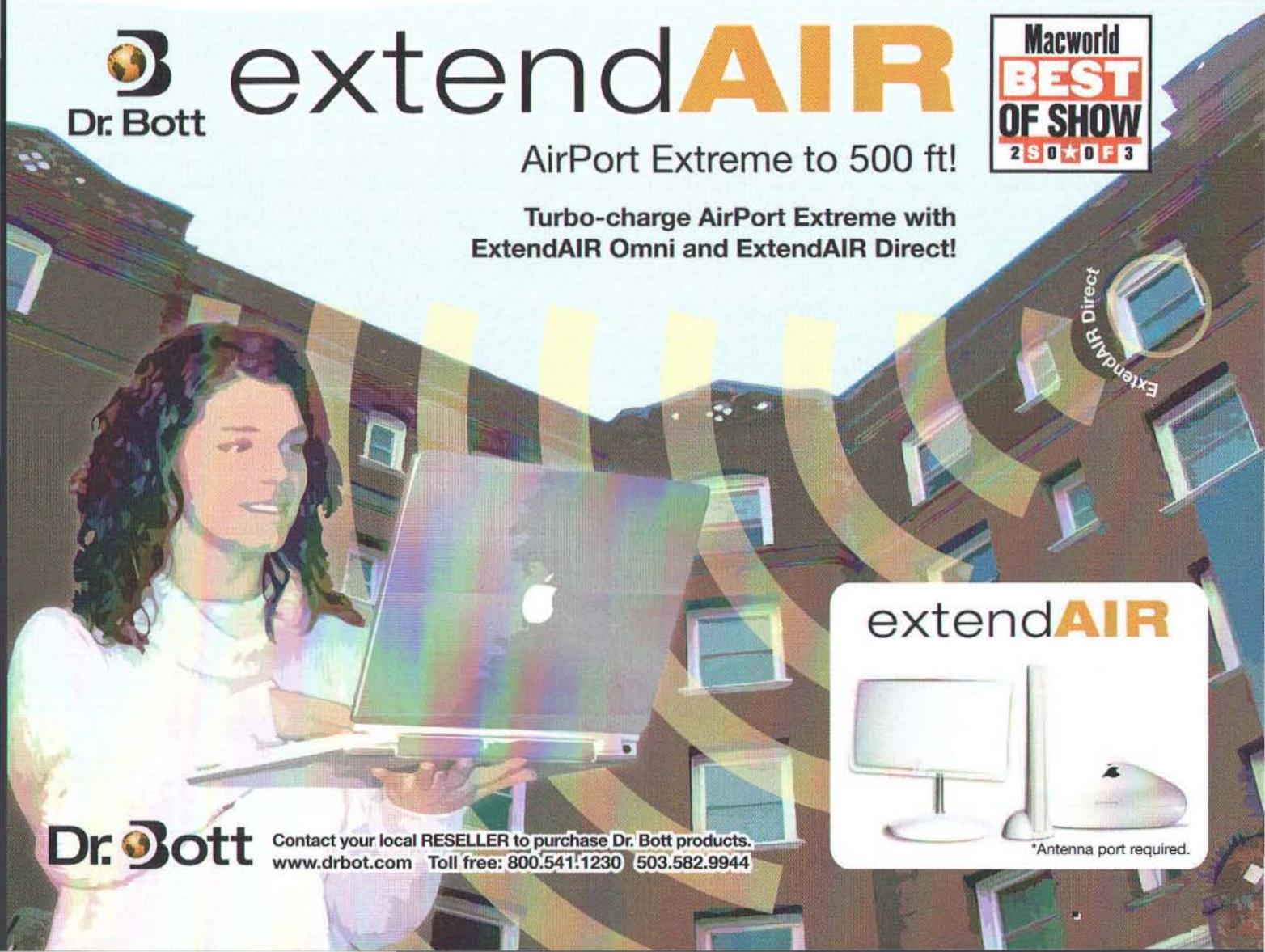
Dr. Bott

extendAIR

AirPort Extreme to 500 ft!

Macworld BEST OF SHOW 2 SOFT 3

Turbo-charge AirPort Extreme with ExtendAIR Omni and ExtendAIR Direct!



Dr. Bott

Contact your local RESELLER to purchase Dr. Bott products.
www.drbot.com Toll free: 800.541.1230 503.582.9944

Vertical Polarization Channel is clear if avg & max < -92dBm

freq Mhz	max dBm	avg dBm	clear	freq Mhz	max dBm	avg dBm	clear
5724	-87	-97	*no*	5788	-93	-99	yes
5728	-81	-95	*no*	5792	-93	-99	yes
5732	-80	-95	*no*	5796	-93	-99	yes
5736	-79	-95	*no*	5800	-93	-99	yes
5740	-79	-95	*no*	5804	-89	-99	*no*
5744	-80	-93	*no*	5808	-88	-97	*no*
5748	-82	-97	*no*	5812	-82	-97	*no*
5752	-91	-99	*no*	5816	-81	-95	*no*
5756	-93	-99	yes	5820	-80	-91	*no*
5760	-93	-99	yes	5824	-65	-80	*no*
5764	-93	-99	yes	5828	-57	-77	*no*
5768	-93	-99	yes	5832	-57	-85	*no*
5772	-93	-99	yes	5836	-57	-83	*no*
5776	-93	-99	yes	5840	-57	-83	*no*
5780	-93	-99	yes	5844	-59	-85	*no*
5784	-93	-99	yes	5848	-63	-79	*no*

Security

With all the talk about security over Wi-Fi, you may be concerned about the security offered by a wireless solution like this. I'm not. First, because these radios are proprietary, it will take a lot more effort for someone to figure them out. Second, the radios will only talk to another radio that it knows the hardware (aka MAC) address of. Third, Trango has built these radios with security in mind. I have a lot greater concerns about other attacks on our network than I do about someone entering in through this wireless entry point.

AppleTalk

If you are still using AppleTalk protocols in your network, no worries. These radios can handle them. Since AppleTalk uses AARP, you will need to turn the multicast packet switch to off (disabled). Once you do that, AppleTalk Phase 1 and 2 pass traffic with no problems. It's pretty cool to pull up a zone at my house from the office ... in fact, I have a Home Automation server at the house that stores my MP3s, and I play them in iTunes over File Sharing on a machine in my office.

Broadcasting Changes

One of the cool things about these radios is that you can have the Access Point broadcast changes to the APs. This is, obviously, especially useful if you have many SUs talking to the AP ... and again, shows the heritage of this radio being targeted for ISPs.

What's an Install Like?

These radios are intended for ISPs looking to deploy in their network. On one hand, that's great because that market won't tolerate a lot of problems ... the radios need to be robust. The flip side to this is that this is not a product that is currently being marketed to the end user. In part, that's because you should have some expertise when deploying it.

Unless you are an expert yourself, you should talk to someone who is. Our experience with Westlink Wireless showed the importance of this. Eddie's ability to size up the network environment and to work out the kinks was important to our install (mostly because we were doing point to point to point instead of point to point as most people would.) For example, I never would have known that it was better to have an AP and SU at David's instead of two AP's.

You need to make sure of several things. First, do you have radio line of sight? From my previous article, radio line of sight takes into account the shape of the zone that the radio waves travel in. This is called the "Freznel" zone. Basically, the radio waves travel between the antennas covering an area that is shaped long an elongated football. In other words, the clearance that you need halfway between the antennas is greater than right at each antenna. In many cases, this means that you are at an advantage if one of the antennae locations is at a higher elevation than the other.

You also need to make sure that you aren't trying to send the signal through or near items that could cause interference. For example, the metal flashings around the top of a chimney might be a problem if you are mounting the radio to a chimney. Or, if you were thinking that you could should the signal through foliage, some may be more tolerant than others. (I learned the hard way that pine trees are really tough to deal with and should always be avoided.)

My recommendation? Either plan for a bunch experimentation and learning, or better yet, buy the radio through a reseller like Westlink Wireless so that you have somewhere to consult if you run into issues.

The Interface

One of the things that I didn't like about the Lucent solution that I had was that it required that I talk to it through a piece of Windows software. VirtualPC, as always, came to the rescue so that I could use my Mac to configure the radios ... and VirtualPC 6 is better than ever in this way.

The Trango radios, though, can be configured not only through a web interface, but also via telnet. A quick overview of the command set gives you an idea of all that you can do with this interface.

Listing 2: Telnet interface for Trango Access Point

```
* ALL
? [command]                                    !
cd [..|main|net|rf|fw|ss]                      date
date <month> <day> <year>                      help [command]
logout                                          opmode [ap]
reboot                                          restart
sysinfo                                         systemsetting
<backup|restore>
sw [<sw #> <on|off>]                           time
time <hour> <min> <sec>
updateflash
<mainimage|fpgaimage|sumainimage|sufpgaimage|systemsetting|sudb
>

* MAIN
bcastscant <all|suid> <ch#> <h|v> [<ch#> <h|v> ...
bcastsuimage <all|suid> <regular|fast> <target hw_ver>
bcastsuimage [stop]                             password
_password <new password> <new password>
set suid <id>                                   set apid <id>
set baseid <id>                                 set ip <ip addr>
set defaultopmode [ap|su] <min,0..60>
set defaultopmode off                           set mir [on|off]
set mir threshold <kbps>
set rssitarget <threshold> <fade margin>
su <suid> [nonstop]                             su all [nonstop]
su changechannel <all|suid> <ch#> <h|v>         su
[live|poweroff|priority]
su <ping|info|status> <suid>                    su reboot <all|suid>
```

```
su restart <all|suid>                    su testrflink
<all|suid> [r]
su powerleveling <all|suid> [target dB]
su sw <suid> <sw #> <on|off>             sudb [dload | view]
sudb add <suid> <pr|reg> <cir,kbps> <mir,kbps> <device id,hex>
sudb delete <suid>                       sudb modify <suid>
<cir|mir> <kbps>
sudb modify <suid> <su2su> <group id,hex>
sudb view

* NET
ping <ip addr>                           tftpd [on|off]

* RF
cf2cf ap [default|<size>]                freq scantable
freq channeltable                        freq writescan [<ch#>
<h|v>] ...
freq writechannel [<ch#> <freq>] ...
freq <ch #> <h|v>                        polar <h|v>
power set <dBm>                          power table
rfreport [reset]                         rfrxthreshold <off|-
80|-75|-70 |-65>
rfrxthreshold table

* FW
mainupdate                               fpgaupdate

* SS
survey <time, sec> <h|v>                 rssi <ch #> <h|v>
apsearch <secs> <ch#> <h|v> [<ch#> <h|v>]...
```

The Good, The Bad, and the Ugly

In general, I love these Trango products. You can't believe how useful it is to change polarity of antennae in software ... or to do a site survey from a telnet interface. More importantly, these are robust radios that are serious business, and I'm thrilled to have them in my network.

There are a couple of things that got to me on these radios, and are worthy of mention here. First, and these are minor, the RJ-45 jack on the power injector is backwards. The clip is against the wall, and it can be very difficult to pull the cord out without having a small flathead screwdriver. Also, the arm that the radio mounts on could be easier to articulate, but it's not bad.

The one thing that I truly despise in these radios is what I can best describe as an over the top security feature. Once set up, you can really only configure these units from one side of the connection because it limits where you can log in from. Remember, these things were designed for ISPs who deploy one or more APs and then have tons of SUs. The idea is that you wouldn't' want a customer to be able to telnet into a SU, so everything needs to happen from the head end side of things. Now, it is possible to log in from the SU side, but you have to do so within the first 30 or so seconds from when it's been power cycled. This is quite the pain and for me, almost useless as I want to manage things remotely, not on site. I have no problem with this feature being available as a software switch, but even if I was an ISP, I wouldn't want this feature turned on by default — I want to be able to configure these things from both sides! And, this one sided nature doesn't stop there, you can't even ping the radio from the client side. This means that an ISP can't even ask their customer if they can ping a radio when the connection is down! What a pain for those in tech support who have chased problems only to find that the client unplugged a cable somewhere. Unfortunately, because it would take a complete rewrite to fix, I don't expect this feature to go away soon.

Another limitation is their use of "opmodes". Radios are usually configured when they are not running in either the Access Point or Subscriber Unit "opmode". Disable the opmode, and you disable the link. Enable the opmode, and you can't configure the unit. This is different from other types of radios that seem to be able to do both.

What's Coming

There are some new things coming down the pike on these radios. A new firmware upgrade will have SNMP capability (this will be a free upgrade). And, there's a dual-band version (5.3/5.8 GHz) coming soon too. Lastly, expected in May 2003, there will be a true point-to-point version of the radios, which may interest many of you.

The End Result

Proof is in the pudding, right?

These radios are fast ... I'm seeing communications at under 300 microseconds (not milliseconds) between the radios. But, a radio ping is different from an IP ping which is what we really care about.

In my point to point to point setup, I came from my older set up doing 1.4 mpbs with 30 millisecond latency to a new set up of 6-8 mbps (depends on the direction) with 5 milliseconds latency. The speed is so different and so much better that, using Retrospect Server, I'm actually now running backups of my home computers to tape drives at the office.

Doesn't get much better than that.

By Ron Davis

Adding Regular Expressions To Your Cocoa Application.

Using MOKit to add the ability to match regular expressions in Cocoa.

Does your application need to parse data out of a bunch of text, or match strings that can vary some, but have a regular syntax? Do you have a Find command in your text editor? If you do you need to add regular expression matching to your app. Regular Expressions are textual representations of strings match pattern. They go beyond just finding a string and let you do things like find a string that begins and ends with certain characters, but can have anything in the middle. Or a string that contain four numbers followed by a letter.

I've been around the Mac a long time and never really thought about grep or regex or other commands that use regular expressions. But OSX changes that. Every UNIX geek out there knows about grep and it various offspring. Scripting languages like Perl use regular expressions as well, so I thought I needed to learn about them. Once I did I was hooked, and wanted to use them in my own applications. That lead me to Mike Ferris' MOKit, a Cocoa framework that lets you easily deal with regular expressions in your application.

Introduction to Regular Expressions

We'll start with a quick look at regular expression syntax for those of you who have no idea what I'm talking about. The introduction will be fast and shallow. If you need more information check out the URL in the Bibliography at the end of the article.

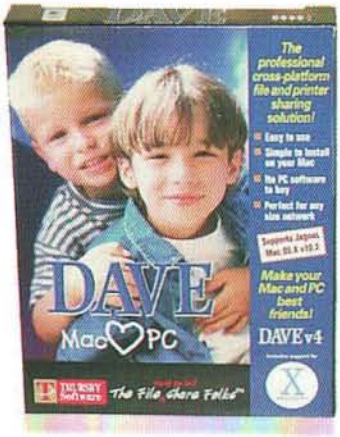
Symbol	Meaning	Example
character	The character typed, with the exception of special characters.	A is a, b is b, etc.
[character – character]	Any of a range of . characters	[a-d] = a,b,c, or d.
.	Period matches any one character, except line breaks.	
#	Matches any digit.	0,1,2,3,4,5,6,7,8,9
\r	return	
\t	tab	
\	The escape character like in printf. Putting a slash in front of a special character allows that character to be matched.	\. matches a period. \\ matches a slash.
?	0 or 1 of the previous . characters	ca?t, matches cat, or ct, but not caat.
*	0 or more of the previous characters	ca*t, matches ct, cat, caat, caaat.
+	1 or more of the previous characters	ca+t, matches cat, caat, caaat, but not ct.
^	any character but the ones after the carat.	(^r23) any character but r, 2, or 3.
pattern | pattern	match pattern **or** pattern.	ca|t, matches ca or t, but not cat.
(pattern)	Matching: treats what is in the parenthesis as a single character. Searching: delineates the information to be remembered in a find.	(ca)*t, matches cat, or cacat, but not ct. c(*?)t, on string coat, returns "oa".

The last pattern there gives you a hint that regular expression can be used in two different ways. One way is matching, where you have a string and you want to know if it is equal to a regular

Ron Davis is a long time Mac programmer, having worked on everything from Virex Anti-Virus to CodeWarrior. His day job is working for Alsoft, and his evening job is R.A.D. Productions, makers of Suck It Down and FinderEye.

With DAVE® your Windows network will welcome Mac users!

*The door is wide open for **sharing files and printers** between Macs and PCs. DAVE offers maximum cross-platform performance and productivity because DAVE:*



- Allows the Mac to be a full partner in a Windows network.
- Supports OS 8.6-9.2x as well as OS X 10.1.5-10.2x.
- Installs only on the Mac...nothing is added to the PC or the server.
- Supports Microsoft standard NTFS file format.
- Provides three options for Security.
- Offers a FREE fully functional demo at www.thursby.com/evaluations.

Download DAVE® TODAY.

www.thursby.com

Macworld 4½

THURSBY Software

expression. This returns a Boolean value, either the string matches or it doesn't. The other way to use regular expressions is to find a substring or strings in a longer string. When you do this you give an expression and you specify what part of the matched string you want back by placing that part in parentheses.

Let's look at an example or two. Say you let the user input a seven digit zip code and you want to make sure they didn't put any letters in there. You could get their input string and compare it against the regular expression "#+", which matches 1 or more digits, but wouldn't match an empty string, nor one with letters in it.

Now say you have an HTML tag for a link like <A HREF=http://www.radproductions.net/>RAD productions</A> and you wanted to pull out the URL. You could search with the regular expression "=(.*?)>" and you would get back http://www.radproductions.net. You may wonder why the ? is there. If you just put ".*", which means match 0 or more characters, you get to the end of the string because quotes and brackets are characters too. This is called a greedy search. Putting the ? tells it to only search until it finds the next part of the expression string.

MOKit

MOKit is a Cocoa framework written by Mike Ferris. It contains some text manipulation classes, one of which handles regular expressions. The underlying regular expression engine is actually a standard package written by Henry Spencer and used in one form or another by a lot of interesting things such as tcl and perl. MOKit classes are "not public domain, but they are free" according to the web page. The code can be downloaded at http://www.lorax.com/FreeStuff/MOKit.html. You can get both compiled frameworks and the source to MOKit. Version 2.6 was used for this article.

MOKit has two main parts, classes for text completion and classes for regular expressions. We'll only be talking about the regular expression classes here. These classes are MORegularExpression and MORegexFormatter. MORegularExpression is the main class for handling the evaluation of regular expressions. It is the one we'll use in our sample code. Here's its declaration.

Listing 1: MORegularExpression interface.

```
@interface MORegularExpression : NSObject <NSCopying, NSCoding>
{
  @private
    NSString *_expressionString;
    NSString *_lastMatch;
    NSRange _lastSubexpressionRanges
                    [MO_REGEXP_MAX_SUBEXPRESSIONS];
    void *_compiledExpression;
    BOOL _ignoreCase;
}

+ (BOOL)validExpressionString:(NSString *)expressionString;

+ (id)regularExpressionWithString:(NSString *)
          expressionString ignoreCase:(BOOL)ignoreCaseFlag;
+ (id)regularExpressionWithString:(NSString *)
          expressionString;

- (id)initWithExpressionString:(NSString *)expressionString
          ignoreCase:(BOOL)ignoreCaseFlag;

- (id)initWithExpressionString:(NSString *)
          expressionString;

- (NSString *)expressionString;

- (BOOL)matchesString:(NSString *)candidate;

- (NSRange)rangeForSubexpressionAtIndex:(unsigned)index
          inString:(NSString *)candidate;

- (NSString *)substringForSubexpressionAtIndex:
          (unsigned)index inString:(NSString *)candidate;

- (NSArray *)subexpressionsForString:(NSString *)candidate;

@end
```

As you can see, it is a fairly simple class. To use a regular expression in your code you create an instance of this class. If you need to keep it around, using the initWithExpressionString methods will probably be easiest. If you're just going to use it in the scope of a single method, use the class methods regularExpressionWithString, so you won't have to deal with releasing. Both of these methods have twins that take an ignoreCase parameter which, if set to YES, will cause evaluations to ignore the case of the characters in the expression and the search string. If you don't explicitly set case sensitivity then searches are case sensitive. Here's an example of how to create an expression for finding HREFs in a string of HTML:

```
MORegularExpression* linkURLExp = [MORegularExpression
                                    regularExpressionWithStrin
                                    g:
                          @"<A HREF=.*?</A>" ignoreCase:YES];
```

If you want to make sure the expression you create is valid you can call the class method validExpressionString, which will return YES if the expression is a valid regular expression. If you want to know what an MORegularExpression object's expression is you can get it from the expressionString accessor.

Now we can actually do some evaluations. As I said previously, there are two ways to use regular expressions, to match a string and to find a sub-string. If you have a string and you want to make sure it conforms to the regular expression you created, you can pass it into matchesString and the result will tell you if it matches. This is what MORegexFormatter does. It is a formatter you can add to a field and it will validate the value in that field by the regular expression you give it.

Getting sub-expressions is interesting. If you just want to find the location in the target string of a sub-string, you can use the rangeForSubexpressionAtIndex method. If you want the whole sub-string back as a new NSString* you use the substringForSubexpressionAtIndex, passing the string you are searching for in the inString parameter. The index is which value in parentheses you want back. There can be 0 to 20 sets of parentheses in a MOKit expression, and the index indicates which one you want the range for. So you could create an expression like "<A HREF=(.*?)>(.*?)</A>" to search for a link in an HTML page. If we used the HTML in Listing 2, and you asked

for index 0 you would get the whole HREF tag: "<A HREF=http://www.radproductions.net/>R.A.D. Productions</A>". If you asked for index 1, you'd get the link back "http://www.radproductions.net/". If you asked for index 2, you'd get back the text "R.A.D. Productions".

Listing 2: Sample HTML

```
<HTML>
<TITLE>R.A.D. Productions Home Page</TITLE>
<BODY>
<A HREF=http://www.radproductions.net/>R.A.D. Productions</A>
</BODY>
</HTML>
```

In a nutshell, that is all there is to finding sub-strings with MORegularExpression. The last method in the interface, **subexpressionsForString**, is there for backwards compatibility and I'm not even going to explain it.

There is one tricky thing about using MORegularExpression in a large amount of text. What happens if you want to find every link in an HTML page? **substringForSubexpressionAtIndex** is only going to return the first occurrence in the string. Turns out there is no way to say, start searching at character n in the candidate string. What I did was truncate the string after each search to find the next one. Here's my code to find all of the links and their URL in an HTML page.

Listing 3: Finding all of the links.

```
-(void)handleHTML:(NSString*)inHTML
{
  MORegularExpression* bothExp =
                [MORegularExpression
                regularExpressionWithString:
                @"<A HREF=(.*?)>(.*?)</A>"
                ignoreCase:YES];

  MORegularExpression* startStopExp =
                [MORegularExpression
                regularExpressionWithString:
                @"<HTML>(.*?)</HTML>"];
  NSString*     result = nil;
  NSRange       range;
  NSString*     curString = [startStopExp
                substringForSubexpressionAtIndex:1
                inString:inHTML];

  do
  {
    range = [bothExp rangeForSubexpressionAtIndex:0
              inString:curString ];
    if ( range.length > 0 )
      {
      NSString*URLString;
      NSString*linkString;
      NSURL*   fullURL;

      result = [linkURLExp
                substringForSubexpressionAtIndex:0
                inString:curString];
      URLString = [bothExp
                substringForSubexpressionAtIndex:1
                inString:curString];
      fullURL = [NSURL URLWithString:URLString
                relativeToURL:baseURL];
      URLString = [fullURL absoluteString];

      linkString = [bothExp
                substringForSubexpressionAtIndex:2
                inString:curString];

      if ( linkString == nil ||
          URLString == nil ||
          ([linkString length]== 0) ||
              ([URLString length]== 0) )
        {} else
        {
        [self addURL:URLString withText:linkString];
        }

      curString = [curString substringFromIndex:
                (range.location + range.length)];
      }
  }
  while ([curString length] > 0 &&
          range.location != NSNotFound );
}
```

A little explanation. The method is in a class that has a method **addURL**. The class also keeps two arrays, one for URLs and one for the link text. When you call **addURL** the URL and the link string are added to the arrays for future reference. The class also knows what the URL of the page you are parsing is, and saves it in a variable called **baseURL**.

The first thing the method does is set up our regular expression for links. Then it makes a new string that will contain only the text between the <HTML> tag. You can use this to limit the search to just a certain part of the page. Then it sets up a loop, which will always execute once and will end when we don't get anything back from our search, or we run out of HTML to parse. Inside the loop we first try to find our expression's range in the HTML. If it isn't there, were done. If we find something, then we use our expression to get the sub-string for the URL. Some times a URL will be relative, so we use NSURL with the page's URL to create a full URL. Then we ask for the second index, which is the link text. If we get both, we add it to our list.

If we find something, then we need to search from the end of the string we found. So we create a sub-string from our current HTML string, that starts at the end of what we found and ends at the end of the current string. This effectively chops off everything from the beginning of the string to the end of what we just found. Then we loop.

Hopefully you've seen the coolness of regular expressions and want to use them in your Cocoa apps. MOKit makes this easy and is easy to use. So go to Mike Ferris' website and download it and add regular expressions to your app.

Bibliography

Mastering Regular Expressions, Jeffrey E. F. Friedl, http://www.ora.com/catalog/regex2/

Using Regular Expressions, Stephen Ramsay, http://etext.lib.virginia.edu/helpsheets/regex.html

Regular Expressions specification, http://www.opengroup.org/onlinepubs/007908799/xbd/re.html

A Tao of Regular Expressions, http://sitescooper.org/tao_regexps.html

BBEdit Grep Tutorial, http://www.anybrowser.org/bbedit/grep.shtml

By Marcelo Amarante Ferreira Gomes, Rio de Janeiro, RJ, Brasil

Introduction to Unix security concepts

Security can only be achieved when you know its basics.

Why Unix?

By now, most, if not all, MacTech readers know that Mac OS X derives from Unix. It may not be *just another* Unix, since it is very different from most other unixes in many aspects. But it definitely *is* Unix. So, if we are to write good software for it, especially if we want that software to be secure, we need to know more than just the basics about the Unix side of Mac OS X.

This is the first article of a series, written with the intent to give you an idea of what computer security really is about, and how to enforce it. We will emphasize the programmer side, hoping to help you write safer applications; but the material in this series will also be of use to system administrators and even power users.

These articles will focus on Mac OS X, since old-timers already know Classic Mac OS, and most newbies are only interested on X. To better explain Mac OS X concepts, though, it is sometimes easier to talk about Unix concepts in general.

There is a lot of historical material in this series of articles. This material is here not only so you can better understand how things evolved and why they are the way they are. It will also to let us learn from the errors of the past and avoid repeating them. You will often see typical attacks crackers used and how security evolved in response to them.

This history-telling approach has the added benefit of passing along a little bit of Unix culture to die-hard Classic Mac OS programmers. In order to write successful Mac OS X software, traditional Unix programmers should learn a bit of Classic Mac OS culture, while traditional Classic Mac OS programmers should have a look into Unix culture. For a discussion on this subject, see (Gomes 2001).

This first article contains no code at all. It will start by defining computer security and then focus on Unix users and groups. You will see how users and groups are implemented in a typical Unix system, how different the Mac OS X implementation is from the typical, and the impact that each of these subjects has on the security of a system.

Defining Computer Security

Unless you're coming from Mars today, you have probably already heard about computer viruses, Trojan horses, DoS (Denial of Service) or DDoS (Distributed DoS) attacks made by crackers against Internet sites, business offices, or even home machines. You might even have already been plagued by some of these. Many people think of these threats whenever they hear the term computer security. While it does include concern about these, the security of a computer system is much more complex than that.

We could define computer security as the task of enforcing just one general rule: Any given user should only be able to use a given computing resource if that user is allowed to. A user, in this sense, could be a real person, a process running in the machine, a library routine, or any entity that can use a computing resource. Let's call this broader concept of user a *cresuser*, for computing resource user. As for the computing resource, it could be memory space, disk space, disk files, CPU time, passwords, processes, or just about anything in the hardware, software or workflow. Even the concepts of *using* or *allowing* might mean different things, depending on context. For instance, using a disk file may mean reading it, appending data to it, modifying data in it, deleting it, renaming it, or yet other creative meanings.

In a multi-user system, security involves the task of making sure that any given person will only be able to do what the system administrator decided that that particular user could do. Examples of such systems include Mac OS 9 with the multiple users feature enabled, Mac OS X, NT-based versions of Windows (NT/2000/XP), other versions of windows with network login enabled, any other flavor of Unix (Unix itself, Linux, [Free|Open|Net]BSD, Solaris, AIX, HP-UX, Irix, etc.), and most mainframe environments.

But even single-user systems have security concerns. Mac OS 8.x and earlier, Windows 9x, aging MS-DOS, dinosaur CP/M, or current ones such as PalmOS and Windows CE are all included in this bundle. To fit them in our definition of computer

Marcelo Amarante Ferreira Gomes is an independent Macintosh, Unix and Internet consultant. He works most of the time with his Brazilian partners, and plays most of the time with his kids and/or his Lombard PowerBook that has the single most important OS in the world, besides most other junk in this industry. You can reach him at `suporte@mac.com`... if you're lucky. :-)

security, you must remember two things. First, even though there's at most one person using the system at any given time, that person is not necessarily allowed to access everything. There may still be the concept of one (or even more than one) administrator and lower-privileged users. And just by adding an interface card and a little software, single-user systems can become part of a larger, multi-user and multi-machine system that we've come to call a network.

The second thing to remember is that our definition of cresuser is much broader than just people; it includes any other entity that could be actively using computer resources. When a new application is installed in a system, it actually becomes a new cresuser in that system. As such, care must be taken so as to limit the reach of the application to only those resources it needs to use. At the same time, we must not impose too restrictive limits; otherwise the application may become so annoying to the person trying to use it, that it effectively becomes useless.

With such a general definition for security, one may get lost while trying to figure out what all of its various meanings might be. And it's also easy to forget to state, or even implement, some of its implications. When we're dealing with computers, there is often more than one way of accomplishing the same task. If a given task is not allowed to a given user, all possible means for that user to accomplish it must be blocked, either explicitly or implicitly.

For instance, I have actually seen a huge security breach in a Unix system shared among students and faculty staff of a certain university. Of course, students were not allowed to browse teacher's files, since the teachers often used that system to write upcoming tests to be applied to those same students. Most security measures were correctly set to block access to directories and files. But they have forgotten about the permission to have raw read access to the hard disk. One student then wrote a program to read the raw information from the disk and interpret the filesystem. That way, he had actually bypassed the Unix kernel filesystem routines, and any security measure built into them, thus getting access to any file in that disk, regardless of the permission bits of those files.

He called his program **thundercat**, a pun on the name of **cat**, the Unix utility to concatenate and list the contents of files. The student got so scared when he realized the meaning of his findings that he never did take advantage of them, and stopped using his program in the same day that he had found an upcoming test. He has never been caught. Before you wonder, I'll assure you: that guy wasn't me.

That episode teaches us a few important lessons.

1) There is often a way of accessing a given resource or accomplishing a given task that you may not have anticipated. Whenever possible, security should be analyzed by more than one person, so that one may see possible weaknesses that the other had overlooked.
2) You don't always need a technical solution to a technical problem. The simplest and perhaps most effective way of solving this particular problem would have been to have separate and isolated (i.e., not sharing the same network) computers for students and faculty staff. That way, students would surely have a harder time trying to access teachers' files, instead of tripping upon them by accident.
3) Conversely, some non-technical problems can actually cause technical ones. This particular university had a periodical computer security check-up procedure, and it was effective. The problem was that it had not been followed. This is more a human resources problem than a technical one.
4) Last but not least, just because you have not *detected* a security problem in your system, it doesn't mean you don't *have* one. Don't ever assume that your system is safe based on the absence of alarms by anti-virus software or other intrusion detection systems. If you feel that something isn't right, trust your instinct, use a little common sense, and investigate.

As a side note to all lessons above, you should be aware that this series of articles concentrates on aspects of security specific to Unix-like systems and their implications on programming. If your particular concerns are more than just a little curiosity, you should read other material that also covers the social side and many other non-technical aspects of security. Good sources are (Schneier 2000) or (Mitnick 2002).

Unix Users, Groups And User Databases

The above definition of security is just too generic. So, let's try to narrow it down to security in a Unix-like system. Before we do, we need a few more concepts. This time, let's use more concrete definitions.

Our concept of cresusers is very broad. We can only talk about cresusers if we have entities to represent them. In any Unix-like system, and that includes Mac OS X, we have users and groups. Neither of these is even close to the concept of cresusers, but they are very important in a Unix environment, especially for those dealing with security. For now, let's study the situation in which cresusers are indeed Unix users, leaving the case of application cresusers, process cresusers, and other types of cresusers for later investigation.

Each Unix user has a name, a numeric user ID and a few other properties, such as the directory they will use by default. By the way, Unix traditionally uses the term *directory* for what we call *folder* in Mac OS parlance. Groups also have names and their own IDs, which may or may not overlap the numeric space of user IDs. As you might guess, a group is a convenient way to refer to more than a single user.

The concepts of users and groups may be implemented in a variety of ways. Traditional unixes have used the **/etc/passwd** file. It is a plain text file with each line representing a user's properties in fields separated by colons, like the one in Listing 1. You may notice that the line for the user **www** in that listing is split. This is in fact just a single line in the file. It appears split only because of magazine format.

The fields included in the /etc/passwd file are, in this order, user name, encrypted password, numeric user ID, numeric group ID, a short description of who or what the user is in real life, the user's default directory and his/her/its default shell. If you don't know what a shell is, just read on. The section *The Unix Command Line* introduces this concept.

Some systems defined a syntax for the user description field, including sub-fields separated by commas or parentheses, but those syntaxes were non-standard. Listing 1 shows an example of this file almost as it has always been. The only exception is that the encrypted password field contains an asterisk for every user.

Listing 1: Typical contents of an /etc/passwd file.

```
##
# User Database
#
# Note that this file is consulted when the system is running in single-user
# mode. At other times this information is handled by lookupd. By default,
# lookupd gets information from NetInfo, so this file will not be consulted
# unless you have changed lookupd's configuration.
##
nobody:*:-2:-2:Unprivileged User:/nohome:/noshell
root:*:0:0:System Administrator:/var/root:/bin/tcsh
daemon:*:1:1:System Services:/var/root:/noshell
www:*:70:70:World Wide Web Server:/Library/WebServer:/noshell
unknown:*:99:99:Unknown User:/nohome:/noshell
marcelo:*:1000:1000:Marcelo Gomes:/home/marcelo:/bin/sh
```

As computing power increased, people started to realize that leaving all encrypted passwords in a single file, accessible by any user was a security risk. Since anyone that could log into the system could read this file, it was easy for a cracker to try passwords at will, or copy it to his own machine and later try to guess each user's password in the comfort of his home. All he had to do was to write a small program to try each password in turn and see if, by encrypting it with the publicly available DES encryption routines, he'd get the same encrypted password that was listed in that /etc/passwd file.

Some programs then became available to automate this process, not only from crackers, but also from system administrators. After realizing the fragility of passwords that users chose, sysadmins periodically ran such programs against dictionaries of common English words to see how easy it was to guess each user's passwords. Some of them even tried permutations of letters in the username and user description. The general idea behind the algorithm used by these programs became known as the *dictionary attack*.

Most dictionary attacker programs were designed to send e-mail to the users that had weak passwords. But since these programs were shared among many people, they eventually ended up in the wrong hands.

Crackers with almost no prior programming experience suddenly found themselves with powerful cracking tools in their hands. Dictionary attackers were not all that hard to tweak, and crackers soon found out how to divert to them the e-mail sent automatically whenever a new password was found.

Mac OS X Killer Tips by Scott Kelby **$19.95** Have you ever had a buddy ask if you "want to know an easy way to do that?" They lean over your keyboard, click a few keys and you learn an undocumented keyboard shortcut, a cool hack, a hidden system option, or just a cleaner way to get the job done. With Mac OS X, most of our favorite tricks are gone – it's a whole new game. This 288 page book is cover to cover tips and tricks for getting the most out of Mac OS X (Jaguar).

Mac OS X Disaster Relief by Ted Landau **$23.95** The unexpected will happen, sometimes you'll need more than a manual. _MacOS X Disaster Relief_ is the first trouble shooting guide to Mac OS X, written by Ted Landau (founder of MacFixIt). This book includes information about Mac OS X's invisible files, third party troubleshooting tools, the smart way to install (or re-install) Mac OS X – plus tips, tricks, and troubleshooting techniques.

Cocoa Programming by Scott Anguish, Erik Buck & Donald Yacktman **$40.95** Over 1200 pages the most comprehensive Mac programming book available for the new Cocoa API. Not for beginners, an essential reference for the serious Mac programmer.

The Wireless Networking Starter Kit by Adam Engst & Glenn Fleishman **$24.95** Practical advice and step-by-step instructions for dozens of common tasks, troubleshooting advice to help you out of sticky situations, and coverage of Mac and Windows implementations of 802.11b, 802.11a, and 802.11g, and other technologies.

Mastering Mac OS X by Todd Stauffer **$26.95** Mac OS X is the most significant change in the Macintosh operating system since day one – you will have questions. Questions about troubleshooting, security, networking, data recovery, fonts, AppleScript, maintenance, or other subjects. There are new technologies you've never seen before, like Darwin, shells, the command line, handwriting recognition, permissions, administration, firewalls, and others. At some point, you will want this reference.

Mac OS X for Unix Geeks by Brian Jepson, Ernest E. Rothman **$16.95** Mac OS X has a BSD core. It's a heady feeling to have the power and flexibility of UNIX on Macintosh, but if you are a UNIX guy you'll find there are some subtle differences. This book is a solid reference, to help you make the switch. It shows you everything from your familiar user shells and directory services to building applications and system management – and how each is done under OS X 10.2 (Jaguar).

Mac OS X Hacks by Rael Dornfest, Kevin Hemenway **$16.95** Mac OS X presents a unique opportunity for combining traditional Unix hacking and Mac OS know-how. This book goes beyond the man pages, pulling the best tips, tricks, and tools from the Mac power users and Unix hackers themselves.

Cocoa Recipes for Mac OS X by Bill Cheeseman **$29.95** A step by step how-to on building an application with Apple's Cocoa framework. The 21 individual recipes, over 750 pages, walk you through the process of building a single, industrial strength application – while illustrating essential topics like handling menus, windows, data storage, user preferences, drag and drop, tables, undo/redo, and more. Want to get started with Cocoa? This is the book.

DEV DEPOT®

www.devdepot.com

877-DEPOT-NOW

When people realized how easy it was to break into systems like that, they thought of changing the users database format. By that time, it was too late to eliminate or change the format of the /etc/passwd file, since way too many applications were using it.

The solution came from noticing that very few programs actually used the password field. So, it could contain any kind of garbage, or even be left blank, and most applications would still work. The few programs that would get broken could later be fixed with little effort.

Enter the concept of shadow password databases. These databases were implementation-dependent, sometimes consisting of binary files, as opposed to the text-only /etc/passwd. Usually named either /etc/shadow or /etc/master.passwd, there was still a text file, very similar to the /etc/passwd file, but containing the real encrypted user passwords. It wasn't readable by anyone, except for the root user. When there were binary files involved, they were compiled from the /etc/master.passwd or /etc/shadow files. The /etc/passwd file was still there, and still readable by anyone, but had the look of listing 1, with no password information.

From then on, programmers were encouraged to get user information from library routines, instead of directly accessing /etc/passwd or any other file. This practice made it possible for system software writers to change once again the format of user or group databases.

Meanwhile, Sun had also developed Yellow Pages, now called NIS, and later came its evolution, NIS+, both implementing network-wide user databases. NIS and NIS+ have become very popular among many other Unix-like OS vendors, besides Sun. NIS seems to have spread a little more, perhaps because it is simpler to implement and has been around longer than NIS+. These centralized network user databases reinforced the idea that programmers should rely on system library routines to obtain user information, instead of going straight to /etc/passwd.

Other centralized users databases, to be used throughout an entire network, have been developed. The most widespread today seems to be LDAP (Lightweight Directory Access Protocol). LDAP interoperates with many different environments: Unix-like, including Mac OS X; Classic Mac OS and even Windows.

Groups can also be defined in many ways, and have traditionally been implemented with a plain text file under the /etc directory, called /etc/group. Nowadays they are generally implemented in the same database that keeps users, be it NIS, NIS+, LDAP or something else.

The traditional /etc/group file was even simpler than /etc/passwd, containing one group per line with colon-separated fields storing the group's name, encrypted password, numeric group ID, and a comma-separated list of users belonging to that group.

Listing 2: Typical contents of an /etc/group file.

```
##
# Group Database
#
# Note that this file is consulted when the system is running in single-user
# mode. At other times this information is handled by lookupd. By default,
# lookupd gets information from NetInfo, so this file will not be consulted
# unless you have changed lookupd's configuration.
##
nobody:*:-1:
wheel:*:0:
daemon:*:1:root
sys:*:3:root
tty:*:4:root
operator:*:5:root,marcelo
mail:*:6:
bin:*:7:
staff:*:20:root,marcelo
guest:*:31:root,marcelo
uucp:*:66:marcelo
www:*:70:
marcelo:*:1000:marcelo
```

It didn't take too long for people to abandon the concept of a group password. It just doesn't work. When the password leaks, nobody takes responsibility. Besides, it was only useful while systems allowed processes to run with the permissions of just a single group. Soon came versions of Unix and Unix-like systems that allowed a single process to have the added permissions of all groups it could belong to, in fact rendering the concept of group passwords useless.

System administrators in charge of the few systems that actually supported group passwords were encouraged to disable it, by changing the encrypted password to an asterisk. The typical /etc/group file ended up looking like listing 2. The group

password field is still there, even in the modern Unix-like systems we have today.

Users And Groups, The Mac OS X Way

Mac OS X has a its own means of storing user and group databases. The /etc/passwd and /etc/master.passwd files are still there, but only for times when network service is unavailable, such as in single-user maintenance mode.

For daily usage, Mac OS X employs a daemon, called lookupd, that in turn gets its information from another daemon, called NetInfo. Unless you fiddle with lookupd's configuration, NetInfo is the definitive source of user information for login and other authentication purposes.

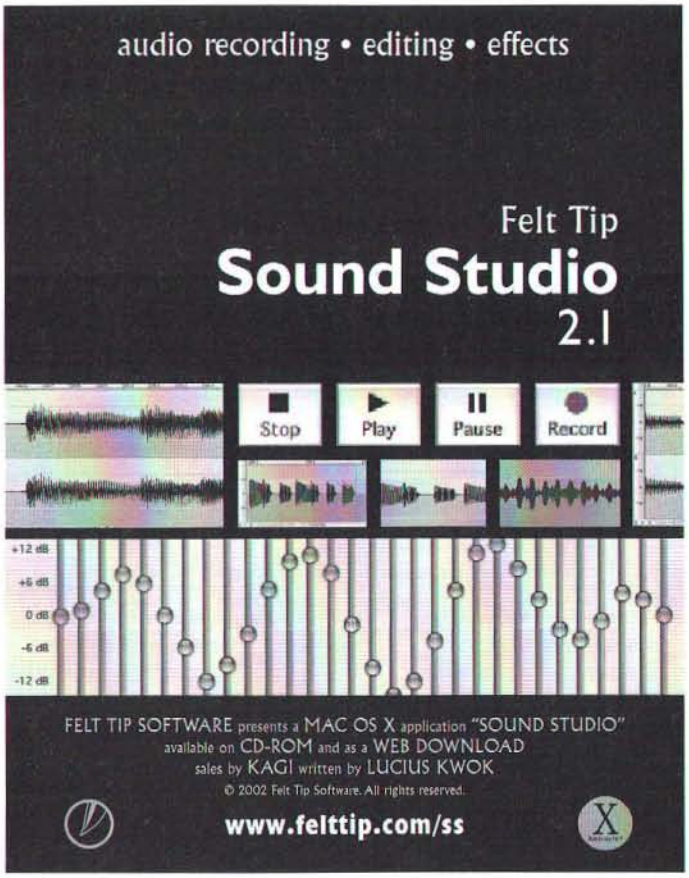
However, Mac OS X may be configured to use NIS/NIS+, as described in (Bresink 2002) for versions prior to 10.2 (pre-Jaguar). As of this writing, in November 2002, the use of NIS with Jaguar is discouraged, but a NIS plug-in for Apple Directory Services is already under development. A similar tweaking may be used to make it look up the needed information in an LDAP database, or even files such as /etc/passwd, although you will probably need additional software to integrate these databases to lookupd and/or NetInfo. Apple says Jaguar has a better integration to LDAP, but I wasn't able to try it out.

Make no mistake, though. NetInfo is much more than a mere users database. It consists of a sophisticated information repository, having a lot in common with LDAP, and is used in Mac OS X to hold lots of other configuration options, besides users and groups. Even information related to a user or a group is not limited to traditional /etc/passwd information. For instance, Mac OS X allows each user to have his or her own *network-shared directory*. The path to that directory is stored in NetInfo, along with all other information about that user.

NetInfo is similar to NIS/NIS+ and LDAP in that they are all capable of maintaining a single database to be shared among many machines. Thus, any user can log into any machine in the network by using his/her own login and password. This won't cause major headaches to the network administrator, since he has a single master users database to take care of.

You can have a better feeling of what NetInfo is by reading (Apple 2001) or the manual page about NetInfo: just type man netinfo at the shell prompt (see the section *The Unix Command Line* below for what a shell prompt is). You could also run the GUI interface to NetInfo, NetInfo Manager. It can be found inside the Utilities sub-folder of your Applications folder. To play it safe, make sure to backup the NetInfo database in /var/db/netinfo/local.nidb and the traditional Unix users database in /etc/passwd and /etc/master.passwd. But keep in mind that the recommended utility for manipulation of user and group information is still the Users control panel.

You should note that NetInfo itself, as well as NIS/NIS+, LDAP or any other network-based users database, may pose a security risk to your system. When the concept of shadow passwords was introduced, the idea was to eliminate the possibility of any user



having access to any other users' passwords, even in encrypted form, to avoid the dictionary attack.

With the introduction of NetInfo, crackers once again have this possibility once they obtain network access to the NetInfo server. I'm sure you will understand why I won't describe here how such access can be obtained and what good this access can do for a cracker. Those of you with more experience in network security probably already have an idea of how it can be done. But I will indeed tell you what measures you can take to minimize your risks.

The choice of strong passwords and good system configuration can greatly reduce this risk. The use of strong passwords, containing non-alphanumeric characters is always recommended, regardless of how you store and manage them. You could even require users to use such passwords if you give them only the choice of changing their passwords through a GUI front-end that checks for the weakness of passwords before committing the change.

A good password, for instance, could be "h4C&3rZ, G0 h0W3!". It has upper and lowercase letters intermixed, numeric and punctuation characters, and has a mnemonic meaning – try to read it as *hackers, go home!* Of course, this one might be difficult for you to remember, but you can probably make up a password like this that you'll be able to memorize. If your memory is really good, you can even use truly random passwords. Just make sure not to write your passwords down, or you'll lose most of the effectiveness of password protection. Another hint: don't use the above password or any other password that has been published. Many people know them, and they will probably make it into many crackers' dictionaries for their next dictionary attack.

Configuration changes may make it harder or impossible for the cracker to get the information he/she wants directly from the server. For instance, to limit the reach of crackers trying to access the database, you may add a firewall to block access to the ports used by NetInfo. Early versions of NetInfo running on NeXT systems used ports 716 through 719. Apple seems to be using 765-768 and 1033 for NetInfo and port 776 for lookupd. You can investigate what ports your system is using by issuing the command netstat -a at the command line of your server. By blocking these ports from the outside, you will limit your headaches to inside crackers. It's always a good idea to limit access to other services as well (in particular, RPC, port 111). If you don't have a firewall, you could install a packet filter a similar blocking mechanism.

If you run a single machine, not meant to be a server, Apple has done its homework and made NetInfo database a bit safer, by setting the trusted_networks property of NetInfo's root directory to an empty value. This should render NetInfo inaccessible from outside machines, although I'd prefer to have the ports blocked. Notice that you can still have local crackers. Anyone that has access to your machine can access NetInfo's database. This is less of a problem, since if any bad guy has access to your machine, the battle is already lost.

The Unix Command Line

Even though Apple makes it hidden from the average user, Mac OS X has a command-line interface. It's in this interface that we find most of the similarities between Mac OS X and other Unix-like systems. In that same Applications folder, you'll find the Terminal application. Fire it up, and let's see how the Unix command line feels.

The prompt that you get comes from another application, your *default shell*, or default command interpreter. The default shell is one of the parameters stored in the user database, and can be different for each user. If you haven't changed any defaults, your commands will be interpreted by /bin/csh, also known as the C shell. The Terminal application just creates one or more windows and acts as an interface between those windows, the keyboard, the shell and you. At the shell prompt, you can type commands to be executed.

You have the option to choose among many shells. Most files under /bin and /usr/bin that end in sh are command interpreters, or shells. Notable exceptions are rsh and ssh, which are meant to establish network connections to contact shells in remote machines.

For instance, try ls -l /bin. This command invokes the ls application, passing it -l and /bin as arguments. ls lists files residing in your filesystem tree. Typically, the entire set of filesystems, the *filesystem tree*, consists of a single HFS+ hard disk volume and possibly a CD or DVD volume. If no argument is given, ls lists only the names of files in the current directory. In this first command, we have specified the /bin directory, so ls will list the files in that directory. The -l argument makes ls output lots of information about the files, instead of just their names. A typical output might look like listing 3.

Listing 3: typical output from the ls command.

```
[localhost:~] marcelo% ls -l /bin
total 8208
-r-xr-xr-x  1 root  wheel      13656 Aug 19  2001 [
-r-xr-xr-x  1 root  wheel      13880 Aug 19  2001 cat
-r-xr-xr-x  1 root  wheel      13764 Aug 19  2001 chmod
-r-xr-xr-x  1 root  wheel      18820 Aug 19  2001 cp
-r-xr-xr-x  2 root  wheel     318108 Aug 19  2001 csh
-r-xr-xr-x  1 root  wheel      14544 Aug 19  2001 date
-r-xr-xr-x  1 root  wheel      26544 Aug 19  2001 dd
-r-xr-sr-x  1 root  operator   18500 Aug 19  2001 df
-r-xr-xr-x  1 root  wheel       9744 Aug 19  2001 domainname
-r-xr-xr-x  1 root  wheel       9184 Aug 19  2001 echo
-r-xr-xr-x  1 root  wheel      60172 Aug 19  2001 ed
-r-xr-xr-x  1 root  wheel      13728 Aug 19  2001 expr
-r-xr-xr-x  1 root  wheel       9396 Aug 19  2001 hostname
-r-xr-xr-x  1 root  wheel      13676 Aug 19  2001 kill
-r-xr-xr-x  1 root  wheel      13604 Aug 19  2001 ln
-r-xr-xr-x  1 root  wheel      26984 Aug 19  2001 ls
-r-xr-xr-x  1 root  wheel      13832 Aug 19  2001 mkdir
-r-xr-xr-x  1 root  wheel      14392 Aug 19  2001 mv
-r-xr-xr-x  1 root  wheel      98704 Aug 19  2001 pax
-r-sr-xr-x  1 root  wheel      35832 Aug 19  2001 ps
-r-xr-xr-x  1 root  wheel       9532 Aug 19  2001 pwd
-r-sr-xr-x  1 root  wheel      24296 Aug 19  2001 rcp
-r-xr-xr-x  1 root  wheel      14292 Aug 19  2001 rm
-r-xr-xr-x  1 root  wheel       9356 Aug 19  2001 rmdir
-r-xr-xr-x  1 root  wheel     465476 Aug 19  2001 sh
-r-xr-xr-x  1 root  wheel       9352 Aug 19  2001 sleep
-r-xr-xr-x  1 root  wheel      23896 Aug 19  2001 stty
-r-xr-xr-x  1 root  wheel       9544 Aug 19  2001 sync
-r-xr-xr-x  2 root  wheel     318108 Aug 19  2001 tcsh
```

```
-r-xr-xr-x  1 root  wheel      13656 Aug 19  2001 test
-r-xr-xr-x  1 root  wheel     465476 Aug 19  2001 zsh
[localhost:~] marcelo%
```

Here we can see the file names at the end of each line. Notice that very long lines, such as the one for the domainname file in listing 3, get broken it up in two. This is done by the routines in the terminal window, and has nothing to do with the ls command. ls does *not* output a line break in the middle of a logical line, no matter how long it is.

The -l flag makes ls list, besides their names, also the type, access permissions, number of links, owner name, group name, size in bytes and creation date and time of the files. In our next article, you will see the meaning of each of these bits of information. All of them have implications on your system's security.

To Be Continued...

We have seen some basics about security in a Unix-like system, focusing mainly on users and groups, and a more generic concept introduced here, that of *cresusers*. Some of these concepts have obvious relations to security, which we've covered here. So that we wouldn't stay only in the theoretical side, we've also covered a little bit about shells and the command-line environment.

In our next article, you will see an in-depth explanation of file access permissions and how they relate to Unix users and groups. Then you'll see some less than obvious security implications of file ownership and permission bits. Later articles in this series will cover practical security measures you could take to avoid common pitfalls when implementing your applications under Mac OS X.

Acknowledgements

Jacques do Prado Brandão always has time and patience to read an article on a subject he does not understand. He helped me correct grammatical glitches and enhance the style of my writing.

On the content side, Marcello, my almost homonymous friend deserves a note here, for being the author of thundercat. The guys that gave him advice on programming and where to find documentation also deserve credit. They know who they are.

I won't discuss the merits of neither my parents nor my wife to have a note here. But I should say that my kids deserve an acknowledgement for always (ok, almost always) be willing to collaborate and give me some silence and peace of mind whenever I needed it. So, here go my thanks to them all.

References

[1] Gomes, Marcelo Amarante Ferreira. "Mac vs. Unix Traditions". *MacTech Magazine* 17:9 (September 2001), pp. 22-27.

[2] Schneier, Bruce. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

[3] Mitnick, Kevin. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, 2002.

[4] Bresink, Marcel. "Integrating Mac OS X in an NIS environment". *Internet* page at <http://www.bresink.de/osx/nis.html>, September 19, 2002 (v 1.91).

[5] Apple Computer. "Security: Mac OS X and UNIX". *Internet* page at <http://developer.apple.com/internet/macosx/securitycompare.html>. The footer states Copyright 2002, but the text states it was written when the latest version of Apache was 1.3.20, that is, in 2001.

[6] NetBSD Documentation <http://www.netbsd.org/Documentation/>.

[7] FreeBSD Documentation Project <http://www.freebsd.org/docproj/>.

[8] The Linux Documentation Project. <http://www.tldp.org/>.

by Tim Monroe

The Sting

Developing QuickTime Applications with "Stinger"

Introduction

In the previous *QuickTime Toolkit* article ("The Vision" in *MacTech*, March 2003), we took a look at developing QuickTime applications using Microsoft's Visual Basic. I noted then that Visual Basic provides no built-in support for displaying or editing or creating QuickTime movies, but that we can work with QuickTime movies using any one of several third-party ActiveX components. Apparently that article struck a nerve somewhere within Microsoft, for several days after the magazine hit the stands, I received a package — postmarked in Redmond, Washington but otherwise with no indication of the sender — containing a single CD. The CD contains what appears to be a pre-release version of a multimedia application framework developed by Microsoft. The product is named "Stinger" and sports a logo of a bee (which is eerily reminiscent of the MSN butterfly).

So far, I've spent only a couple of days working with Stinger, but it's clearly a software development environment of some importance for QuickTime developers. It allows us to write applications that support the standard litany of Microsoft proprietary media and file formats (AVI, WMV7, ASF); Stinger also supports MPEG-4 audio and video formats. This already is pretty surprising, as it represents an abrupt shift on Microsoft's behalf toward open standards for multimedia content. What's even more surprising is that we can use Stinger to develop applications that can create, open, and display QuickTime movies. And the list of operating systems that can run Stinger-developed applications is truly amazing; Stinger can create multimedia applications that run on Windows, Mac OS, several flavors of Linux, Palm OS, and even MS-DOS itself! For instance, **Figure 1** shows the QuickTime sample movie displayed by an application running under Red Hat Linux; note the distinctive Red Hat "Bluecurve" graphical interface.

Figure 1: *A QuickTime movie playing in Red Hat Linux*

In this article, we're going to take a preliminary and unfortunately brief look at Stinger. I should say in advance that the information here is derived mainly from my own tinkering around with it (there was no documentation on the CD), and some of it is pure conjecture. Also, the name "Stinger" is quite likely a code name — though judging from the professional finish to the logo, the final product name will probably have some apiary connection ("Visual Beesic" perhaps?).

Microsoft and QuickTime

Perhaps the most amazing thing about Stinger is that it exists at all. I noted in the previous article that "Microsoft...does not have a history of promoting QuickTime as a multimedia creation or delivery technology". That was an understatement. In fact,

Tim Monroe in a member of the QuickTime engineering team. You can contact him at monroe@apple.com. The views expressed here are not necessarily shared by his employer.

Multiple formats. Multiple platforms. Complex installers.

Aladdin solves the compression and installation puzzle.

Trying to figure out how to handle multiple compression formats and platforms?
The StuffIt Engine solves the compression puzzle.

Aladdin's StuffIt Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the StuffIt file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low as $99/year

To learn more, visit: www.stuffit.com/sdk/

StuffIt Engine SDK™
The power of StuffIt in your software.

Looking for the easiest and fastest way to build an installer?
StuffIt InstallerMaker completes your puzzle.

It's not enough just to write solid code anymore. You still have to write an installer for your users. StuffIt InstallerMaker makes it simple and effective.

- StuffIt InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with StuffIt InstallerMaker.

Prices start at $250

To learn more, visit: www.stuffit.com/installermaker/

StuffIt InstallerMaker™
The complete installation solution.™

Aladdin Systems

www.stuffit.com
(831) 761-6200

Microsoft has a history of trying to derail the adoption of QuickTime, at least on Windows operating systems. Here is an excerpt from the court's decision in the recent Justice Department anti-trust case against Microsoft:

> "§. 105. Beginning in the spring of 1997 and continuing into the summer of 1998, Microsoft tried to persuade Apple to stop producing a Windows 95 version of its multimedia playback software, which presented developers of multimedia content with alternatives to Microsoft's multimedia APIs. If Apple acceded to the proposal, Microsoft executives said, Microsoft would not enter the authoring business and would instead assist Apple in developing and selling tools for developers writing multimedia content....Apple would have been permitted, without hindrance, to market a media player that would run on top of DirectX. But...Apple's QuickTime shell would not have exposed platform-level APIs to developers. Microsoft executives acknowledged to Apple their doubts that a firm could make a successful business out of marketing such a shell." (From the ***Court's Findings of Fact***, Civil Action No. 98-1232.)

At one meeting in April 1997, Apple engineer Peter Hoddie asked incredulously: "Are you asking us to kill playback? Are you asking us to knife the baby?" According to court testimony, a Microsoft official responded: "Yes, we want you to knife the baby." Ouch. (Insider trivia factoid: this exchange prompted a member of the QuickTime engineering team to print up tee-shirts bearing the slogan "Don't Knife the Baby.")

It gets worse. Some early versions of Microsoft's multimedia playback application, the *Windows Media Player* (also known affectionately and perhaps too accurately as *WiMP*), had a tendency to hijack the QuickTime movie MIME type (that is, "video/quicktime"). When a Windows user would navigate to a web site containing embedded QuickTime files (with the file extension .mov), the movie files would be opened by WiMP, not by the QuickTime plug-in. For most media types, and particularly for QuickTime VR movies, this would result in a less than optimal experience.

Further, Internet Explorer version 6.0 (which was first shipped with Windows XP) and version 5.5 with Service Pack 2 applied broke browser-based QuickTime movie playback altogether. That was because those versions of IE stopped supporting NetScape-style plug-ins and because the QuickTime plug-in was implemented as a NetScape-style plug-in. Indeed, it was precisely to address this issue that Apple developed (with Microsoft's assistance, it must be pointed out) the Apple QuickTime ActiveX control that we investigated in the previous article.

I could continue down this road *ad nauseum*, but the fact is that I come to praise Microsoft, not to bury it. After all, as I've suggested, Stinger represents a complete reversal of this "knife the baby" strategy. So, in quasi-Watergate style, let's follow the honey.

Stinger Overview

The CD I received contains two folders, one labeled "Runtime Installers" and the other labeled "DevStudio Installers". The first folder contains what appear to be runtime environment installers for the target platforms (more on this later). The second folder contains only a single executable file, StngrInstl.exe. After running the application, I launched Microsoft Visual C++ Developer Studio and was greeted with the "Tip of the Day" shown in **Figure 2**.

Figure 2: *A Visual Studio Tip of the Day*

Dutifully, I selected the New menu item in the File menu. Sure enough, the list of available projects included "QuickTime Application" and "QuickTime Component", as shown in **Figure 3**. Clicking on "QuickTime Application" revealed that Stinger can generate QuickTime-savvy applications for seven different platforms: Windows, Mac OS 9, Mac OS X, Palm OS, Red Hat Linux, Mandrake Linux, and MS-DOS. I was, to put it mildly, intrigued.

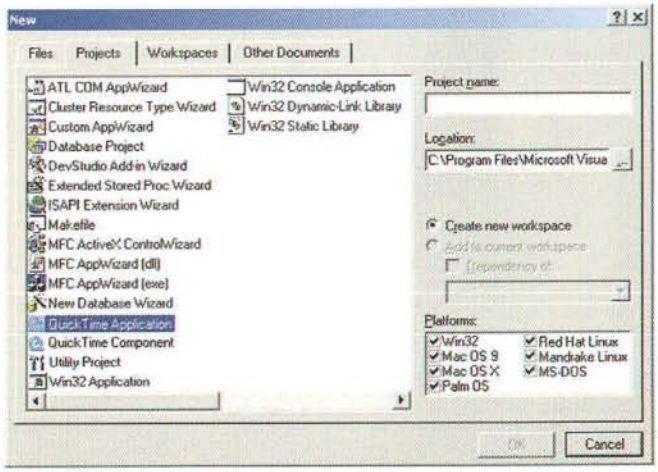


Figure 3: *The New Project dialog box*

I named the new project "StingerShell". Recall that our goal in this current series of articles is to replicate the functionality of

our existing application QTShell. On Windows, QTShell provides support for opening multiple QuickTime movie files and displaying them in child windows inside an enclosing frame window. In other words, QTShell on Windows uses the Multiple Document Interface (MDI). It turns out that the default applications created by Stinger use the Single Document Interface (SDI): they open QuickTime movies in a window whose content area completely contains the movie and the movie controller bar. (The size of the movie within the content area can be adjusted, but for present purposes we don't need to do so.) So the only thing we need to do to bring the default Windows application into line with an SDI version of QTShell is add the Test menu — where we add our application-specific items — and add some code to handle those menu items.

Adding Menu Items

The programming model supported by Stinger is unexciting: it's straight C code that calls Windows APIs for basic application services (event handling, message processing, and so forth) and QuickTime APIs for multimedia services. **Listing 1** shows the code that is called when the user selects a movie in the file-opening dialog box.

Listing 1: Opening a QuickTime movie file

OpenMovie

```
void OpenMovie (HWND hwnd, char *szFileName)
{
    short    nFileRefNum = 0;
    FSSpec   fss;

    NativePathNameToFSSpec(szFileName, &fss, 0);
    SetGWorld((CGrafPtr)GetNativeWindowPort(hwnd), NULL);
    OpenMovieFile(&fss, &nFileRefNum, fsRdPerm);
    NewMovieFromFile(&sMovie, nFileRefNum, NULL, NULL,
          newMovieActive, NULL);
    CloseMovieFile(nFileRefNum);
}
```

The only interesting departure from the standard Windows programming model that we've worked with throughout this series of articles concerns the specification of the application's resources. The default project contains a file StingerShell.sc that defines the application's menus, icons, text strings, and dialog box layouts. This .sc file is analogous to the .rc files we've worked with previously, but supports additional resource compiler directives necessary to support deployment on different platforms. **Listing 2** shows how we can add in the Test menu but — for Palm applications only — exclude the Help menu.

Listing 2: Specifying the application's menus and menu items

StingerShell.sc

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N",        ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",    ID_FILE_OPEN
    MENUITEM "&Close\tCtrl+W",      ID_FILE_CLOSE
    MENUITEM SEPARATOR
```

```
    MENUITEM "&Save\tCtrl+S",          ID_FILE_SAVE
    MENUITEM "Save &As...",            ID_FILE_SAVE_AS
    MENUITEM SEPARATOR
    MENUITEM "E&xit\tCtrl+Q",          ID_APP_EXIT
  END
  POPUP "&Edit"
  BEGIN
    MENUITEM "&Undo\tCtrl+Z",          ID_EDIT_UNDO
    MENUITEM SEPARATOR
    MENUITEM "Cu&t\tCtrl+X",         ID_EDIT_CUT
    MENUITEM "&Copy\tCtrl+C",          ID_EDIT_COPY
    MENUITEM "&Paste\tCtrl+V",       ID_EDIT_PASTE
    MENUITEM "C&lear",                 ID_EDIT_CLEAR
    MENUITEM SEPARATOR
    MENUITEM "Select &All\tCtrl+A",ID_EDIT_SELECTALL
  END
  POPUP "&Test"
  BEGIN
    MENUITEM "&Hide Controller Bar\tCtrl+1",
                                       ID_TEST_HIDE_CTRL
    MENUITEM "&Hide Speaker Button\tCtrl+2",
                                       ID_TEST_HIDE_SPKR
  END
#if !TARGET_OS_PALM
  POPUP "&Help"
  BEGIN
    MENUITEM "&About StingerShell...",  ID_APP_ABOUT
  END
#endif
END
```

The code required to handle our application-specific menu items is, once again, completely unexciting. I'll leave it as an exercise for the interested reader to dredge up a past issue of *MacTech* to see how it works.

Building and Running the Applications

Now let's build the applications. Select "Build StingerShell" in the Build menu (or press the F7 key). Once the build process completes, the Debug folder will contain the usual collection of object files and other junk; it will also contain debug builds of StingerShell for the target platforms. If we launch the Windows application StingerShell.exe, we'll see the error window shown in **Figure 4**.

Figure 4: *A runtime error message*

This is where the installers in the "Runtime Installers" folder come into play. Let's run the installer WinHiveInstl.exe. Once we've done this, we can try launching StingerShell.exe again. This time, everything proceeds as expected. The movie playback window is shown in **Figure 5**.

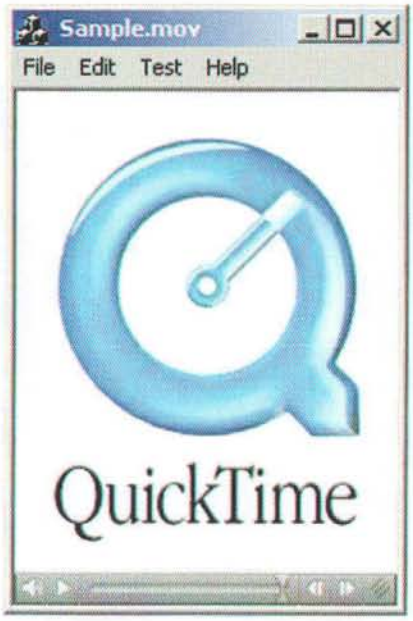


Figure 5: *A QuickTime movie playing under Windows*

We can transfer the other compiled applications (and runtime installers) to their target operating systems. The file StingerShell.app is a Carbon application that can run on Mac OS 9 or Mac OS X. **Figure 6** shows StingerShell running under Mac OS X.

Figure 6: *A QuickTime movie playing under Mac OS X*

The file StingerShell.rhl is the executable for Red Hat Linux (see **Figure 1** again). The file StingerShell.ml is the executable for Mandrake Linux (**Figure 7**).

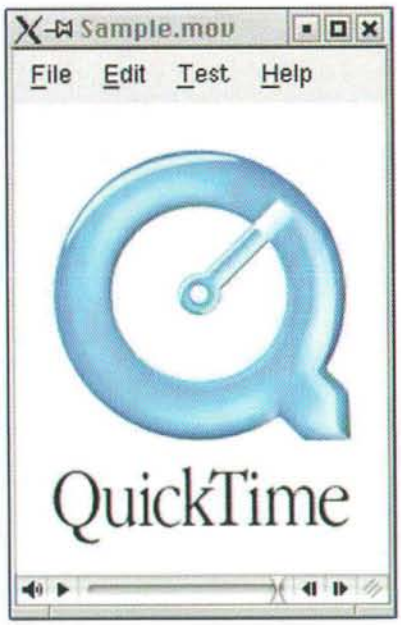


Figure 7: *A QuickTime movie playing under Mandrake Linux*

Apparently there is already a PalmOS application with the trademarked name StingerShell, so I've renamed the Palm application QTShell.prc. When we download the executable to a Palm device, we'll see it in the list of available programs, as shown in **Figure 8**.

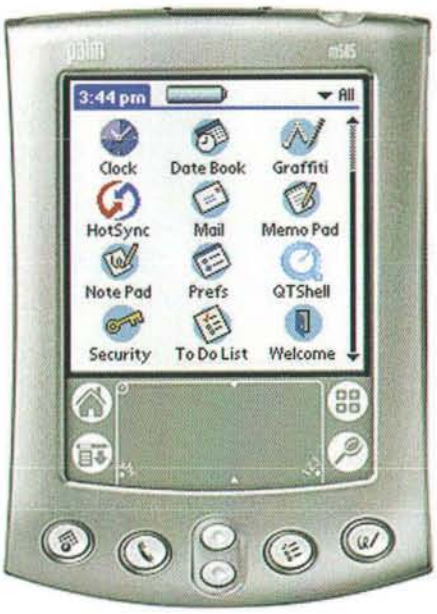


Figure 8: *QTShell in the list of Palm programs*

Figure 9 shows QTShell running on the Palm device.

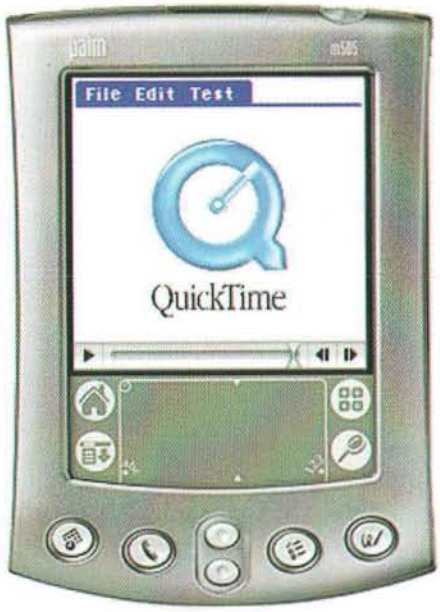


Figure 9: *A QuickTime movie playing under PalmOS*

Conclusion

This is all very cool, but how does it work? Apple provides QuickTime runtime support only for Macintosh and Windows operating systems. How can we run QuickTime applications on Linux and Palm systems as well? This is where I'm reduced to pure speculation. As we've seen, our compiled applications would not run without the prior installation of the so-called ".HIVE" runtime environment. I'm guessing that this is some sort of multimedia extension to Microsoft's .NET environment. Apparently Microsoft has shoehorned the QuickTime libraries into the platform-specific .HIVE interpreters and also provided some sort of emulation layer for those platforms that do not natively support QuickTime. Or so I'm guessing. If I had more time and a good debugger, I could probably figure out more of what's happening under the hood.

No matter *how* it works, it's clear that indeed it *does* work. It's sad to realize that Microsoft has beaten Apple to the punch in supporting QuickTime movie playback on Palm devices and on Linux platforms. The performance is not good (I was able to achieve a playback rate of only about 0.5 frames per second on a Palm device, and only about 2.5 fps under the Linux systems), but the API coverage appears to be fairly complete. Sigh.

Credits and Retractions

Special thanks are due to an unknown source within Microsoft for providing a pre-release copy of Stinger.

You may have noticed that *QuickTime Toolkit* articles sport titles borrowed from movies (appropriate for a series on QuickTime, eh?). Perhaps a better title for the present article would have been the 1969 Jack Lemmon and Catherine Deneuve comedy, "The April Fools".

Free Mac® OS X!

... now, for a limited time, when you buy MYOB AccountEdge 3, we'll give you Mac® OS X absolutely FREE.

Evolving with the Mac since 1989

Small Business Management and Accounting

www.myob.com/us MYOB **800-322-MYOB (6962)**

Small Business. Smart Solutions.

Offer details available online at www.myob.com/us. Valid for retail product #meurm purchased between 1/7/03 and 3/31/03. Cannot be combined with other offers. Void where prohibited. MYOB, the MYOB logo and AccountEdge are registered trademarks of MYOB Ltd. Mac and Mac OS X are trademarks of Apple Computer registered in the U.S. and other countries. © MYOB 2003

REVIEW

By Rich Morin

Books on Cocoa Programming

The field is filling in...

I've spent the last few months learning my way around Cocoa programming, using a combination of Objective-C and Perl. I'm not anything like a wizard yet, but I'm getting by and even starting to feel a bit confident. I have received lots of support from folks on assorted mailing lists, but these books got heavy usage first. "Read the Fine Manual", and all :-).

Whether you're just starting up the Cocoa learning curve or well along the way, I'd suggest that you consider getting some of these books:

Building Cocoa Applications: A Step-by-Step Guide
Simson Garfinkel & Michael K. Mahoney
O'Reilly, 2002; ISBN 0-596-00235-1
622 pp.; $44.99

Cocoa Programming
Scott Anguish, Erik M. Buck, Donald A. Yacktman
SAMS, 2002; ISBN 0-672-32230-7
1245 pp.; $59.99

Cocoa Programming for Dummies
Erick Tejkowski
Wiley, 2003; ISBN 0-7645-2613-8
366 pp.; $24.99

Cocoa Programming for Mac OS X
Aaron Hillegass
Addison-Wesley, 2002; ISBN 0-201-72683-1
383 pp.; $44.99

Cocoa Recipes for Mac OS X: The Vermont Recipes
Bill Cheeseman
Peachpit Press, 2003; ISBN 0-201-87801-1
752 pp.; $44.99

Learning Cocoa with Objective-C (2nd edition)
James Duncan Davidson & Apple
O'Reilly, 2002; ISBN 0-596-00301-3
360 pp.; $34.95

Executive Summary

If you're just starting out, I'd recommend "Cocoa Programming for Dummies". Really. It's economical, a fast read, and will give you a quick and painless introduction to the main topics you'll be studying later.

Next, you'll want to skim the two O'Reilly books ("Learning Cocoa with Objective-C" and "Building Cocoa Applications"). You'll also want to keep them close at hand for reference. If you already have some O'Reilly books, you'll know what to expect from these: clean, solid presentation, consistent organization, good production values, etc..

With this background, you'll be ready to tackle any of the last three volumes. I found each of these to contained things that the others did not, but "Cocoa Programming" was by far the most detailed and definitive of the lot. In fact, if you can only afford to buy one Cocoa programming book, "Cocoa Programming" would be the best use of your money.

Cocoa Programming for Dummies

The goals and approach of this book are about as far from "Cocoa Programming" as it is possible to be. It isn't particularly detailed, let alone definitive, and screenshots fill many of the pages. Nonetheless, it is a worthwhile book. In fact, it's the best "jump start" book I've seen, providing a painless way for "newbies" to pick up the key concepts of Cocoa programming. Just be ready to buy some more books before trying to shift into second :-).

Although the book itself contains relatively little code, the supporting web site (www.dummies.com/extras) has quite a bit. In fact, it provides a 16 MB ".sea" archive with source code, pre-built packages, and more. The examples are all simple, but that's quite appropriate to the introductory nature of the book.

Learning Cocoa with Objective-C

Although I like this book, I was surprised by the number of typos and other small errors I encountered while reading it. I shouldn't be able to find dozens of glitches in any O'Reilly book; given that this is a second edition, I should be hard-pressed to find any at all.

Nonetheless, it's a good introductory book. It covers the basics of Cocoa, Objective-C, OOP, etc.. It won't take you very far into Cocoa programming, but it's a fine lead-in and will also be handy to keep around as a reference. Like "Cocoa Programming for Mac OS X", it contains a wealth of useful diagrams, which frequently help to clarify the topic under discussion.

Building Cocoa Applications

Although I found this book useful, it is not my idea of a "Step-by-Step Guide". After short introductions to Aqua and Apple's development tools (~60 pages each), it jumps headlong into three large "Applications":

- "Calculator: Building a Simple Application (152 pp.)
- "MathPaper: A Multiple-Document, Multiprocess Application (154 pp.)
- "GraphPaper: A Multithreaded, Mouse-Tracking Application (150 pp.)

Each application is used as a vehicle for explaining certain aspects of Cocoa programming. For instance, the Calculator app is used to introduce "Nibs and Icons", "Delegation and Resizing", "Events and Responders", and "Darwin and the Window Server".

Cocoa Programming for Mac OS X

This book "splits the difference" between "Cocoa Programming" and "Cocoa Programming for Dummies". It moves a lot faster than one and a lot slower than the other. It has a nice sprinkling of screenshots, many useful diagrams, and a fair amount of detail. I'm particularly happy with the coverage of design techniques, programming pitfalls, and other "incidental" issues.

The writing style is quite informal and relaxed, but the content is very solid. If you can't afford to attend a course at "Big Nerd Ranch" (the author's training facility), you may find that this book provides a similar set of information, delivered in a similar manner.

Cocoa Recipes for Mac OS X

This book is based on the "Vermont Recipes" application, a "sampler" which shows off most of the simpler Cocoa widgets (e.g., buttons, drawers, sliders, tabs, and text items), complete with error-checking code and other niceties. , tab views, and drawers). It walks the reader, in painstaking detail, through the steps needed to reproduce and enhance the application.

The web site (www.stepwise.com/Articles/VermontRecipes) contains both the app and the project files that were used to build it. If you like to work your way through someone else's examples, this site (and the accompanying book) should be very worthwhile. Alternatively, the app provides a fast way to play around with a lot of Cocoa widgets.

Because my own approach involves blundering my way through my own projects, I haven't used this book (or any of these books, really) as the authors intended. Nonetheless, I was able to use it from time to time as a way to clarify obscure points of Cocoa programming and tool use.

Cocoa Programming

As noted above, this is a very detailed and definitive volume. In fact, I found it rather intimidating at first: lots of text, a fair amount of code, and only a small number of screenshots and tables. With time, however, I realized that this book was answering my programming questions better than any of the other volumes.

I like this book so well, in fact, that I'd like to see it expanded into two volumes. The first could be augmented with more introductory information (e.g., on Apple's developer tools, the Cocoa programming model, etc.) The second could concentrate on some of the more esoteric aspects of the Apple Frameworks.

Coming Attractions

Other Cocoa programming books are on the way. Some of these will fill in holes left by the current crop; others will attempt to supplant existing books. None of the current books cover Cocoa programming in languages other than Objective-C (e.g., AppleScript, Java, Perl, Python, Ruby, and Visual Basic); I know of at least one such book (on Perl) that is in development.

I would really like to see a detailed reference for the entire range of Apple's developer tools. I have spent far too much time trying to find information on Interface Builder and Project Builder; I'd love to have a single book that covers them, as well as some of the obscure tools in "/Developer/Tools" and "/Developer/Applications", in painstaking detail.

By John C. Welch

Jaguar for Administrators

Where does Jaguar stand on the network?

Jaguar from the IT point of view

Writing a review of an operating system from the network administrator's point of view is always a bit odd. Especially when that operating system is Mac OS X. For one thing, most of the features in Mac OS X that other reviews center on don't really matter here. Aqua, haxies, and other user gadgets aren't that important. Rather, the deep esoteric parts of the OS get a rare moment in the limelight. It also means that the review is far more concerned with how Mac OS X plays with other environments than you would normally see. Now, this is not going to be a feature by feature review, but rather an overview of Mac OS X and Mac OS X Server from a network administrator point of view. With that in mind, onward and downward.

Mac OS X is an odd beast of an operating system for network administrators, especially those used to traditional Unix operating systems. There are a number of differences that will trip you up. Configuring Apache, for one. Other things like CGI access and SSL have rather unique configuration issues. Keep in mind that just because you know BSD and/or other flavors of Unix inside and out doesn't mean you have the same level of knowledge in Mac OS X. In other words, don't assume.

Once Mac OS X is set up, it's a great client, but getting there is painful. If you are not running Mac OS X Server, then creating things like groups and aliases is far more onerous a task than it ought to be. The only way is to either use NetInfo Manager, which is still not reliable enough to be completely trustworthy, or use command line utilities, such as niutil, and nicl. If you want to import a group of users, the command line utilities are the only reliable way to do this without Mac OS X Server.

"... is far harder without Mac OS X Server" is a recurring theme when administering Mac OS X. The only way Apple supports running Mac OS X networks is with Mac OS X Server, and as a result, trying to run Mac OS X *without* Mac OS X Server, while certainly doable, is a lot more tedious than it should be.

Open Directory

One of the big reasons for the tediousness in managing Mac OS X is Netinfo. While it was groundbreaking when it was introduced with NeXTSTEP, it does poorly compared to other directory systems like LDAP, Active Directory, and NDS. It's poorly documented, doesn't integrate will with other tools, and is for all practical purposes a single platform system. NetInfo Manager is, compared to any other directory management application, painful (a directory management tool that doesn't have online help is a bad thing). While NetInfo is certainly not as fragile or obtuse as the Windows Registry, it's not much better. The tools for dealing with NetInfo are (if possible) even worse than those for the Windows Registry.

Even on Mac OS X Server, there is no good tool for managing the directory as a thing. You can manage aspects of the directory, like users, groups, machines, printing, etc., but those are things you use with the directory. The only tool available to manage the directory itself, is NetInfo Manager. Compared to other directory tools, NetInfo Manager is lousy at managing NetInfo Directories and domains. If you are trying to implement and manage LDAP domains under Mac OS X Server, it's even worse. NetInfo was simply neglected for too long, and is too isolated to be as central as it is to an operating system trying to move into the future. Apple needs to put NetInfo in the same coffin as Mac OS 9, and the sooner the better. They need to get a real directory management tool while they're at it, too.

Since we're on the topic of directories and directory integration, we need to touch on what Open Directory actually is and is not. First off, it's not a specific directory type, although it's focused on LDAP. Open Directory is more of an architecture that allows different directory types to plug into the Open Directory structure. Mac OS X ships with a number of Open Directory plugins, including LDAPv2, LDAPv3 (new in Jaguar), and NetInfo. Note that the plugins don't have to be 'directories'. Rendezvous, AppleTalk, SLP, and SMB are also part of the Open Directory container. If you think of Open Directory as a container for pouring different ways to manage Mac OS X on a network, then it gets less confusing. Third parties can also write plugins for the Open Directory architecture. Thursby Systems DAVE 4 is one example.

John Welch <jwelch@mit.edu> is an IT consultant for MIT Central IS, and the Chief Know-It-All for TackyShirt. He has over fifteen years of experience at making computers work. John specializes in figuring out ways in which to make the Mac do what nobody thinks it can, showing that the Mac is the superior administrative platform, and teaching others how to use it in interesting, if sometimes frightening ways. He also does things that don't involve computertry on occasion, or at least that's the rumor.

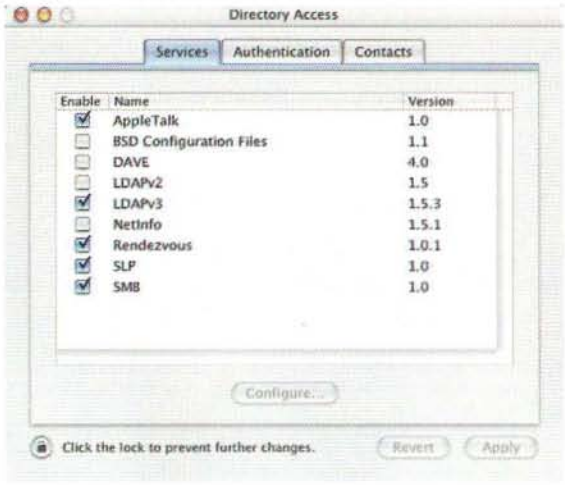


Figure 1: *Directory Access showing Open Directory plugins*

As we see in the illustration above, the settings for various Open Directory plugins are configured via the Directory Access application, found in /Applications/Utilities. This brings me to one of the biggest complaints I have with Mac OS X and Mac OS X Server as a network administrator, and that is documentation. While I can easily download gigabytes of information on every development API on the platform, here is the sum total of Directory Access' help on setting up your system to work with Active Directory:

If you want a Mac OS X computer to get administrative data from an Active Directory server, the data must exist on the Active Directory server in the format required by Mac OS X. You may need to add, modify, or reorganize data on the Active Directory server. You must make the necessary modifications by using tools on the Active Directory server.

We all understand that Apple has limited resources compared to some, but for Mac administrators, getting their Windows counterparts to work with them is hard enough; forcing them to tell a Windows administrator that the only way to get the Macs to integrate properly requires schema modifications, or third party products on the Active Directory server is just not going to fly. In the networking world, Apple is the little guy. They need to make the extra effort. The same applies to other directory schemes. While the support for more 'standard' LDAP schemes, such as OpenLDAP and Sun's Directory Server, is better, Apple still needs to make this as simple as selecting the Directory Server type, clicking "Add my machine to this domain", and entering an administrative password.

This is not to say that Open Directory is worthless. Quite the contrary. By including proper support for things like LDAPv3, LDAPv3 over SSL, and support for using Berkeley configuration files across the network, Apple has made Mac OS X's directory services support work on the level that we need it to. The LDAPv3 support in Open Directory is well thought out, and answers most, if not all of the problems with LDAP support in Mac OS X prior to Jaguar. Things like using

SSL to encrypt the data flow, using a specific distinguished name and password when connecting to the LDAP server, and being able to set custom mappings (even writing them to the server from the client if need be) all show the hard work that Apple has put into the plumbing of Open Directory. The implementation is still not there yet. A great deal of the benefits of LDAP are lost as soon as they hit NetInfo, and its rather poor implementation.

There's no AppleScript support in Directory Access, or indeed, any of the Mac OS X Server tools from Apple, so you can't automate any of this setup. If you want to set up a hundred new Macs, there's a lot more manual work than there ought to be. Being able to automate client setup is a critical need for any administrator, and for Mac administrators that means AppleScript. Learning shell should not be necessary to run a Mac.

Unix tools

Since Mac OS X and Mac OS X Server are based on FreeBSD, how do they stack up as Unix operating systems? The answer is both "really good" and "really annoying". For the most part, an experienced Unix administrator will be quite at home with Mac OS X. They will find themselves stumbling a bit as they hit differences, however. User home directories are not where they 'should' be. The default file system isn't UFS. There are no tape drivers, so you can't just use tar, gnutar, dump, etc., to do quick and dirty backups. Dump would have major issue with HFS+ anyway.

There's also no command line toolset that makes doing things via SSH a bit easier than running command after command. One of the best examples of this is SMIT, found on IBM's AIX. It runs in both X11 mode, and command line mode. SMIT is really nothing more than an application that takes input, and uses that input to run various administrative related shell commands. But even over the command line, it gives you a basic menu driven method for getting work done. Thanks to the command line mode, SMIT works quite well over dialup, something that Apple's GUI tools do not do well at all. As Chuck Goolsbee, ListMom for the Mac-Mgrs list says, "SMIT Rocks!".

Apple has done a *much* better job of making sure there are command line versions of the GUI utilities in Jaguar, especially for things like remote setup, remote network setup, software update, remote installations, and others. This was a critical issue for many administrators, as there was simply no way to migrate to Mac OS X if the only way to do things was via a GUI. (Of course, prior to Jaguar, you couldn't use Mac OS X Server to manage Mac OS X boxes, so that was a problem that Jaguar alleviated.) We all understand that to most users, the command line is anathema,. Even some Mac administrators have opined that Apple shouldn't have released Mac OS X without a GUI equivalent for every single command line utility. This is just ridiculous. While the command line should *never* be a requirement for users, administrators fall under different rules. For administrators,

the command line is a force multiplier that they have needed for quite some time.

While Mac OS X currently doesn't ship with an X11 implementation, the recent release of the X11 preview at Macworld Expo shows that Apple is taking this environment seriously. No, X11 is not Aqua, nor is it as easy to use as Aqua. It is, however, a standard GUI environment available for every Unix in use, and thanks to things like GNOME, and KDE, X11 is getting close to being as easy to use as Aqua. In addition, there are a number of critical applications like MatLab and IDL that will probably never show up as full Aqua applications, but rather as X11 applications. Other Unix applications, such as Open Office, are likely to show up first as X11 applications, with Aqua versions coming later. With that in mind, it is far more intelligent for Apple to have an X11 implementation that is easy to use, configure, and integrates properly with elements like Aqua. We would like to see Apple take this even further, and integrate X11 into Mac OS X to the extent that there is no difference, *to the user,* between an X11 application and a 'true' Mac OS X application. If Apple can get X11 to a 'double click and go' state, then they will have answered a huge complaint from the Unix world.

Server Administration Tools

This area of OS X has probably seen more improvement under Jaguar than almost any other area of the OS. With Jaguar Server, Apple *finally* shipped a server that had the tools to properly administer Mac OS X. While there has been a great deal of public comment about Quark slowing Mac OS X adoption, the lack of tools to run Mac OS X networks prior to Jaguar Server hurt that adoption rate in places with multiple Macs as much, if not more than any one application.

Again, while Apple provides you with solid GUI tools to run your server and your network, they are marred by the same stumbling blocks that crop up all over Mac OS X. For example, you can use Kerberos to authenticate clients connecting via FTP and AFP, but not SMB or NFS. There is one click support for AppleTalk over SSH, but nothing else, even though FTP and NFS both benefit greatly from additional security. Apple's GUI tools for Windows file sharing are so limited that you should essentially ignore them, and use SWAT, an open source, web based Samba configuration tool instead. Even with SWAT, Apple's modifications to Samba are still hit and a miss. They hit by integrating Samba and Open Directory, but miss by making it painful, and in some cases impossible, to use the more interesting features of Samba, like its ability to be a Windows NT 4 Primary Domain Controller. Once again, it's that last 20% of the effort that makes a product great, instead of merely adequate.

Apple's provided mail server, while much improved over earlier versions, is still frustrating when you compare it to other products such as Communigate Pro. There are no easy ways to enable functions like IMAP or POP over SSL in the UI. On the other hand, enabling Kerberos authentication is a single checkbox away. Of course, setting up Mac OS X Server as a KDC, heck setting up Kerberos servers in general is not easy.

Apple's implementation of Apache is the same mix of ease of setup, and frustration at the hands of odd implementations. Setting up CGI access for user sites on a Mac OS X Server box is more work than it should be, and completely different than almost every other Unix platform. But then again, you find that setting up SSL is as easy as can be, although SSL access is a less common need than CGI access. You can enable WebDAV for the server as a whole, but you have to then enable it individually for each site. It is not clear that you have to do so, an issue that bites Unix administrators a lot, especially if Mac OS X Server is the 'new' OS for them.

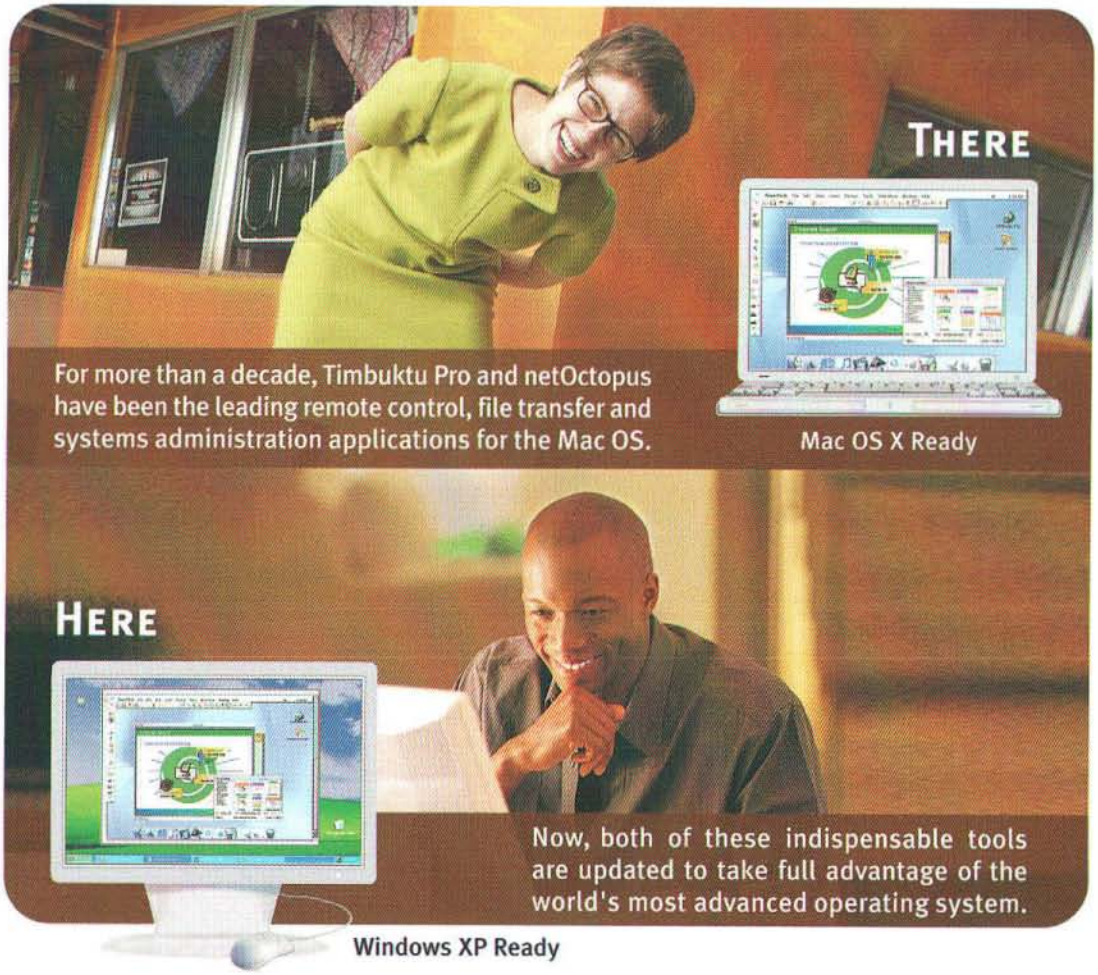
This occurs across the board for many of Apple's tools. Firewall setup is easy, yet Apple just throws it's hands up at DNS, only allowing you to turn DNS on and off from the GUI tools. The rest has to be done via config files and the command line. Considering that Apple's server tools will not work, or work very badly if DNS isn't set up correctly, this service should have a *far* more comprehensive set of tools. Certainly more than just an on/off switch. Apple's server monitoring and status tools give you an excellent display of hardware and service conditions, but the only effective tools to kill a runaway process are ssh, top, and kill from the command line if you aren't at the system console.

Conclusion

Mac OS X and Mac OS X Server are still in fairly early stages of development, and that needs be taken into consideration when reading the negative comments above. On the whole, they are both excellent tools for administrators. However, the continuous stream of stumbling blocks thrown in an administrator's path makes that excellence hard to see at times. What needs to be done is a matter of focusing. Apple needs to start making each minor update a chance to focus on one set of problems, fix them, and move on to the next. Each update is still trying to do too much at once. As a result, you end up with minor improvements and glaring omissions (like the XServe drive bay door problem, *still* not fixed as of Mac OS X 10.2.3). Administrators would be better served by an update that only deals with Apache issues, or Samba issues than by one that fixes one or two issues each from a half dozen categories. The administration tools need to be taken to that next step as well. Windows file sharing and DNS administration tools lack of functionality must be fixed and soon. The lack of a good directory administration tool needs to be remedied as close to yesterday as possible. Finally, NetInfo needs to go away. It was a great product in its day, but the fact that it was neglected for so long, and the practical limitations of NetInfo on Mac OS X are two problems that are not going away, and are not worth fixing, when you have LDAP being used globally, and continually improved.

So, from the administrators viewpoint, Mac OS X is about 80% of the way to being a truly first class client and server operating system. Once that work is done, then Mac OS X will truly be the superior OS that it strives to be.

Stay In Control Wherever You Go.



Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.
http://www.timbuktupro.com

netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.
http://www.netoctopus.com

Learn more, try it, or buy it online. Call us at 1-800-485-5741.

timbuktu® • netOctopus®

netopia.

By Cabel Sasser

Redesigning Right

How we adapted the Transmit interface for Mac OS X

They'll Notice... Oh, They'll Notice

Mac users are crazy. Not clucking-like-a-chicken-in-the-reference-stacks-of-the-public-library crazy, but the good kind of crazy – the kind that can inspire an intense passion, love, commitment and evangelical fervor for something as dry as a computing platform. And as Mac users, we love our interface in an almost unnatural way – your application could print diamonds and candy from an ordinary bubblejet while whistling Dixie (and I'd like a copy, please), but if your app isn't easy, pretty, and thoroughly Mac-like, it will be dismissed. With the arrival of Mac OS X, the importance of interface has become even more critical – Apple has set the bar for quality UI design even higher.

Thus, if your Mac OS X application at all looks *not quite right*, you're in trouble. Maybe your default buttons are aligned left, or your status text is still Geneva 9, or your Preferences are only accessible via the "Edit" menu, or the background of your window is filled with a scaled photograph of a beautiful Icelandic mountain goat. No matter what the offense, when a Mac application doesn't follow the Apple Human Interface Guidelines to the letter, *people notice,* even if they have no idea that there is such as a thing as "Interface Guidelines".

When we took our first stab at a Mac OS X port of our FTP client Transmit (the Carbon-based 1.7b2), we pretty much brought the interface over identically – the good ol' "Bare Minimum" port. We updated the font to Lucida Grande, replaced some old folder icons, spaced things out a little, and added a new 128 x 128 icon, but that was about it. After a few months of getting under Mac OS X's skin, it was clear to us that 1.7b2 wasn't even remotely enough. Mac OS X threw almost everything we knew about Macintosh interface design out the window, but things had improved – now, when we booted in Mac OS 9 things felt odd and cumbersome, no longer the other way around.

So, once we made the decision to rewrite Transmit in Cocoa, we decided we would remodel the entire interface as well: the goal, as with all Panic products, was to make Transmit feel like a part of Mac OS X itself, as if it came straight out of Cupertino.

The Interface

Two things make Transmit unique among FTP clients. First, the main interface is all-in-one – it not only contains its own built-in local file browser, but also an in-line connection panel for entering server information, all in one window. This unification allows the user to breeze through their FTP session. Secondly, we rely heavily on Finder-like metaphors and behaviors to make our application feel more like an extension to the Finder itself rather than a stand-alone FTP app. For example, you drag and drop to upload and download. You click and wait to rename. You drag to the trash to delete. You drag into a folder to move. As we like to say, "If you can use a Mac, you can use Transmit."

When updating the main interface for Mac OS X, it was important that we maintained a careful balance: adopt the best new design features that Mac OS X has to offer, but don't destroy the essence of our application's identity – don't break what works well, but improve what doesn't. It was a very tough task. Here's the result:

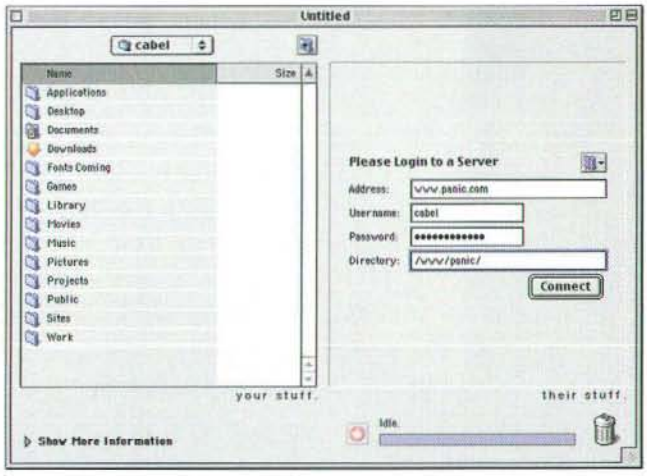


***Figure 1**. Mac OS 9's Transmit 1.7*

Cabel Sasser is the other co-founder of Panic Inc., and is Panic's primary right brain, responsible for design in all its many forms. He also buys all the weird new food products for the guys in the office to try out (Funky Fries, anyone?). Feel free to send Cabel any e-mail at cabel@panic.com.

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

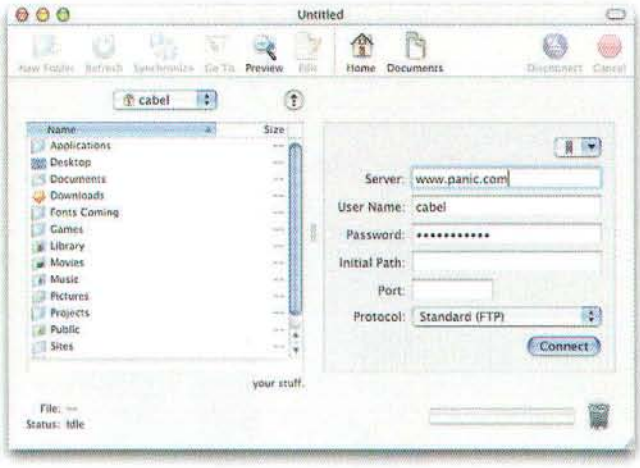


Figure 2. *Mac OS X's Transmit 2.0*

Consider Toolbars

The first thing we added to Transmit 2's interface was a toolbar. You'll notice them in virtually every Mac OS X application, from Mail all the way down to the humble Print Center. Despite seeming gimmicky, we've learned to love toolbars: it's far easier to mouse over to a shiny icon than to navigate through menus when you want to perform an action in a hurry. Additionally, some of your best features can get lost under heaps of cascading menus — toolbars help new users easily discover all your application has to offer.

So, if your application has a lot of "actions" tucked away in the menus, there's no quicker way to make it feel like a good Mac OS X citizen than to install a toolbar. Here are some things we kept in mind when designing the Transmit 2 toolbar:

- Choose your items carefully. Don't make everything available in your toolbar, only the actions you think the user is likely to access often. If the feature is so obscure that only 5% of your users care, it probably doesn't need to go into the toolbar. For example, we didn't add "Send Raw FTP Command" to the toolbar – it's so hard core, it's best left in the menus.
- Choose your default toolbar set even more carefully. Don't overwhelm the user with a toolbar longer than the Great Wall of China on first launch. In Transmit, we have "Upload" and "Download" buttons that the user can add to the toolbar if they desire, but we don't put them in the toolbar by default – we want the user to discover the "drag and drop" method of uploading and downloading first, since it's far more natural.
- Don't add fundamental application actions to your toolbar. It may seem obvious, but your toolbar should not have a "Quit" button. The same goes for "Preferences", "About",

"Help", "Hide", etc. Your application should be already providing a common and consistent place for these typical, every-app actions.

- Try to use the toolbar to simplify your interface. You'll notice we moved the "Cancel" button from our main interface window up into the toolbar – nestled among the other actions, it reduces the clutter in the main interface view, and fits well with other actions.
- Think a bit about your default toolbar's item order. In Transmit's case, the two destructive actions – Cancel and Disconnect – are the black sheep of the toolbar, shunned away on the right and separated with a flexible space to reduce the likelihood of accidental clicks. You'll also notice that a separator sits between the action icons (New Folder, Refresh) and the location icons (Home, Documents), letting the user know that these sets of icons behave slightly differently. Simple things like spacers and separators can do wonders for usability.

Figure 3. *Transmit's default toolbar.*

Rework the Interface

With the toolbars set, we started picking and choosing our primary interface elements, carefully considering the usefulness of each and asking ourselves, "Can we do this better?"

In the bottom of the original Transmit window was a "Show More Information" disclosure triangle. When clicked, the window would resize to reveal multiple informational text fields containing data about the active connection. Although it was a fun widget to mindlessly click on while a transfer is running, there was really no reason for "More Information" to be optional – virtually everyone ran Transmit with "Show More Information" open. After all, who wants to choose "I want to know less"? We took the contents of this optional panel and, although we lost a little information in the transition, we managed to roll all of it into a single easy to understand status line in Transmit 2.

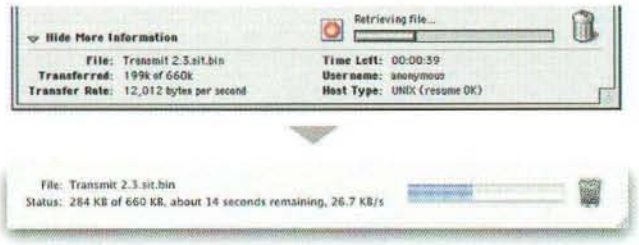


Figure 4. *The optimization of "More Information"*

We kept the trashcan in the lower-right hand corner of Transmit's interface, but it is still a matter of some contention. Optimally, since Mac OS X offers an always-available trashcan in the Dock, we wanted to encourage users to drag their items to the Dock instead, removing an extra control from our window in the process. However, we found that old-school Transmit users felt very lost and bewildered without their convenient trashcan waiting for them in the corner, and users who had their Dock set to auto-hide were even more confused about how to delete items, since no trashcan was visible to them while using Transmit. In short, we let the trash can stay... for now.

Finally, you might have noticed a path pop-up control, and a parent button, smack dab in the main interface. "Aren't we supposed to 'simplify' these kinds of things by placing such items in the toolbar?", you might be asking. Well, technically, you're right. After all, the Finder has both a back button and a path pop-up in its toolbar, and they work just fine.

The problem here, however, is that Transmit has two separate file views in one window to consider: local and remote. If the "parent" button was in the toolbar, the user would first have to click the side they wanted to change — say, "your stuff" — and then, with a side selected, they'd be able to click the parent button. By having two sets of these controls, one on each side, the user can immediately affect the one side they wish to affect — we eliminate a selection step in a frequently used action.

Avoid Metal

There's one very important thing Transmit's interface isn't: metal. With the introduction of Jaguar, Apple provided a simple checkbox in Interface Builder that still to this day gives me night tremors and cold sweats: "Textured Window". This innocent looking box makes your application all shiny and metal, very much like iTunes, iPhoto, and, somewhat questionably, Address Book.

In the Aqua Human Interface Guidelines, Apple provides a very logical description of when the metal look is acceptable. You should read it if you're curious, they've certainly put some good thought into it. But I will personally summarize it thusly: "Your application should use the textured appearance [...] never."

In all seriousness, no Mac user wants a patchwork-quilt of a desktop, with some windows ice-lined and others molten-forged. Unless your application fits the acceptable guidelines to the letter, I say don't check the box. I know it looks cool and has a high novelty value, but if you ship a word-processor with the metal texture, we will come to your house and hurt you. Seriously.

The Preferences

Ah, preferences – gateway for the everyday tweaker, long-standing domain of the power-user, and defender of all-that-can-be-changed. In migrating an application from Mac OS 9 to Mac OS X, you'll likely spend a great deal of time re-creating your preferences, since they're very control and layout heavy. The Mac OS 9 Human Interface Guidelines had a very strict set of rules for typical window layout (lots of group boxes, left-justified controls, tabs galore, and the like), and all of it is irrelevant in Mac OS X.

When it comes to re-designing your preferences, I say: start from scratch! Don't load up your old resources and excitedly start re-arranging and re-sizing them. Instead, open up Text Edit, and write out a good old-fashioned simple list of each of your Preferences. Then, walk through the list, and consider each preference carefully — look for ways to combine preferences, to clarify controls, or weed out preferences that are so obscure that they're probably not worth the clutter. Finally, once your list is done, fire up Interface Builder, and start building a new Preferences window from scratch, working only from your list. Your preferences will thank you for it! (Although that'd be a little creepy...)

Here's some other advice we took when bringing Transmit's preferences up to code:

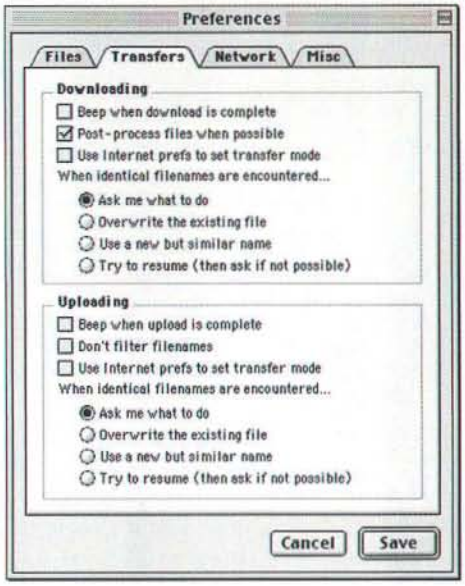


Figure 5. *The "Transfers" preferences from Mac OS 9's Transmit 1.7*

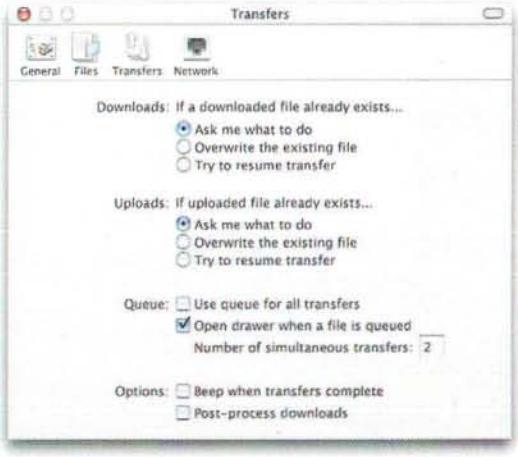
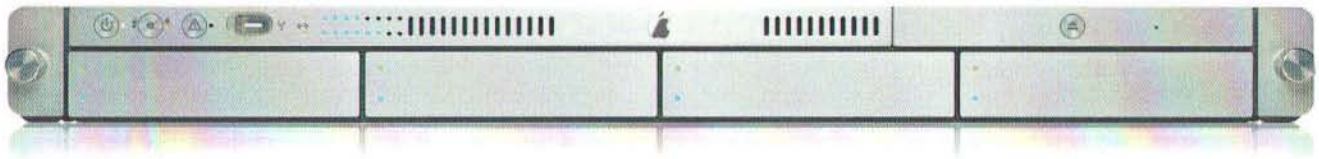


Figure 6. *The "Transfers" preferences, all nice and Mac OS X-like. Ahhh.*

Xserve



Density optimized rack mounted Mac OS X Server

UNIX-based Server Solutions from Apple

Nearly half a terabyte of storage per 1U

630 gigaflops of processing power

Hot-Swappable drives

Industry-standard 1U rack-optimized design

There has never been a better time to buy...

Small Dog Electronics carries factory refurbished models too offered at substantial savings!

(subject to availability)

Small Dog Electronics

www.smalldog.com

1673 Main Street, Route 100, Waitsfield, Vermont

Apple Specialist

To learn more: http://www.smalldog.com/xserve/

- Think centered. Mac OS 9 dialogs and windows were traditionally left justified, but in Mac OS X, the world is centered – even the drop shadows cast down from top and center. That doesn't mean each item should be individually centered, though — instead, group your related controls and captions, and then center that entire group. See our "Downloads:" group for an example.
- Keep a close eye on fonts. As of this writing, Lucida Grande reigns supreme, almost always in 12 point. And take heed: there should be no Geneva, Charcoal or, god forbid, Chicago!
- Let your window breathe. Mac OS X windows and dialogs are much more spaced-out than their Mac OS 9 counterparts, and the result is a much more airy, much less totallysupercrampacked experience. Now that the minimum resolution supported by Mac OS X is 800 x 600, there's no reason to jam all of your preferences into a 100 x 200 window.
- Get rid of group boxes. They're oh-so-1997. Place a static text heading to the left and top of a series of controls if you wish to group them visually. Or, for bigger groups, use a static text with a horizontal rule bottom-justified to the right of it, then place the controls (and sub-groups) underneath.
- Remove your tabs. True Mac OS X preferences should now use a toolbar-like control at the top of the window, and the window should resize appropriately when sections are accessed.
- Prune. Like I said, we took this re-writing opportunity as a chance to re-consider every preference we had. "How many people use this?" and "Is this really that important?" should be questions you ask frequently. If you remove a control that 4 people use, sure, you'll probably hear from those 4 people (it might be the same 4 people we always hear from!), but hundreds of other people might just be more likely to find what they're looking for now that the obscure option has gone away. I know it's counter-intuitive to take something **out** of your application, but be brave! And if you need moral support, call us.

Follow these basic rules – and keep the HIG on your nightstand for a while – and you'll be well on your way to a shiny, Mac OS X-like preferences experience in no time.

The Icons

In the land of Mac OS X, icons are king. Really. There's no greater investment you can make for your application than going out, finding a quality icon designer, and making an original set of icons. Cobbling together your own icons from other applications, snagging icons from the Finder, or re-using freeware icon sets is almost always a bad idea – this usually lends to situations such as: "Well, I can't really find an icon for 'Reset Cache' anywhere, so I'll use this cool icon of a rubber duck in a top hat wearing a delightful little monocle, since people love rubber ducks! And monocles! Right? Besides, help tags were invented for a reason, right?"

In the case of Transmit, we went straight to The Iconfactory, who we've worked with since 1997 – we're always amazed by their creativity and quality. They're not alone, though – there are many talented Mac OS X-specializing icon artists, some who may even work for free (although, if you work for Microsoft, I suspect they might charge you.)

Your best bet is to Google for something like "Mac OS X Icons" or search through the news links at The Iconfactory, browse the free icons out there, find an artist whose style you like, and write him or her an e-mail explaining that you need their talents! Make sure the artist is equally good at both the tough-to-clarify 32 x 32 toolbar icons *and* the very-large and photorealistic 128 x 128 application icons. Also, make sure you can clearly explain your exact icon needs. Good direction is crucial! And when it comes down to price (always tricky), I suggest simply asking the author what they want in return for the work. Icon people are good people, as long as you're a good person in return. (I guess that's true for many things in life.)

Toolbar Icons

When working on the Transmit 2 toolbar icons, we kept coming back to the importance of consistency – the icons needed to resonate with each other, in order for the application to feel cohesive and whole. At the same time, we wanted the icons to feel Finder-like. So, we had to strike a delicate balance between our own style and the Finder's style. The complete results are as follows:

Figure 7. *The many toolbar adventures of Transmit 2.*

As you can (hopefully) see, each icon shares the same language: badges are consistently white shapes with a soft shadow, non-real-world graphics are rendered in graphite, each bookmark-related icon shares the same basic book, etc. As a whole, the Transmit toolbar icons feel like a family, and help the application gel.

Don't be afraid to continually revise your icons, either, as your application winds its way through development. Our toolbar icons went through many revisions before we felt fully comfortable with them. It's an inexact science – my only advice is to trust your intuition and tweak until the icons just feel right. If you know for a fact you have no design intuition, grab the pickiest Mac-using friend you know and run everything by them. (As for our many stabs at the "refresh" icon, pictured below, I'd like to thank Dave at Iconfactory for being so patient when I wrote him notes like

"I don't know... that one's a little bit too wispy." Great, Cabel. Thanks.)

Figure 8. *Six – yes six! – revisions of Refresh.*

There was one snag during the creation of our toolbar icons that is worth mentioning. Since many features in Transmit mirror those in the Finder, our instinct was to make our toolbar icons as Apple-like as possible. We even considered using the exact same icons that the Finder uses for things such as "New Folder". However, as we began to investigate Apple's icon language, we noticed some inconsistencies, and eventually came to the conclusion that it would be smarter to invent our own, Apple-like style, but with a Panic twist.

Take, for example, icon badges – little icons that combine with other icons to create new icons. Phew! Here are three of Apple's:

Figure 9. *Some of Apple's different badge styles*

As you can see, each icon has a different take on badges – one is pseudo-3D, one is centered flat white, one is white with a border, etc. With so many styles present in Apple's work, we struggled to pick which one was "correct" for our own badges. It was a riddle with no answer, as even Apple is still refining and perfecting their style — there is no "bible" just yet. So, we wound up creating our own New Folder icon, based on Apple's, but fitting within our own Transmit style. It's a bit smaller, and uses a badge style consistent with all of our other icons. We think it's a good balance.

Figure 10. *Transmit's "New Folder" (left), inspired by Apple's.*

Application Icons

I've always believed that an application icon needs to have this unspoken attractiveness to it that just makes you want to uncontrollably double click it even when you don't have to. I'm not sure if this is vestigial stimulus from early childhood or some kind of terrible comment on society as a whole, but I think the more attractive, cute, and polished an icon is, the more likely it will gain a permanent place in a user's dock, and thus be a part of their everyday toolkit. Admittedly, I've put some ugly icons in my dock in the past, but only out of necessity – once a more attractive icon came along, I've almost always dumped the old one and immediately upgraded to the new, beautiful one. Okay, now this is just sounds weirdly personal.

Figure 11. *Our little guy is all grown up! (From l-r: Mac OS 7.5, 8.1, 10.0, and 10.2)*

The very first Transmit icon used the standard limited color palette for Mac OS 7.5 icons, and was basically a "Copland-style" three-dimensional truck. The next revision of the icon got a whole lot of shading and a little bit of an alpha-channel shadow, thanks to the 32-bit icon capabilities of Mac OS 8. Once Mac OS X came out, the truck grew to 128 x 128 – but it was one of The Iconfactory's first Mac OS X icons, so it was really more of a "scaled-up" 32 x 32 illustrative truck, rather than a photorealistic, Mac OS X-style icon. Finally, the Transmit 2 icon underwent the ultimate upgrade — a true piece of icon art, even down to the headlights and tire treads, that fits perfectly in Mac OS X.

When designing your application icon, don't forget to spend a few days doing what we call "The Dock Test". That's when you drag your new icon into your dock and.. well.. leave it there for a while! Just live with it. Does it fit in? Does it stand out? How does it look next to everything else? The Dock Test is terribly helpful in deciding if your icon is ready to go.

Finally, in case you've ever wondered "why a truck?" Well, Transmit was originally called Transit! It made a lot more sense back then, really, before the lawyers got involved.

...AND THE REST

The HIG, impressive tome that it is, can't always give you all the answers. In our case, there was at least one place where we "flew solo", creating a control that isn't really covered in the HIG or used anywhere else in Mac OS X.

The favorites menu in Transmit sits in the corner of the connection panel, containing a list of the user's favorite servers. In Transmit 2, it's a pop-up menu with no caption to the left, and the pop-up menu itself doesn't display any text – just an icon. Yes, it's not something you see often, and those are certainly two violated rules.

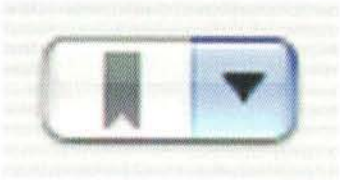


Figure 12. *Transmit's unique "favorites" control.*

But, in our defense, we think this custom control works pretty well by combining two global concepts. First, a pop-up menu, distinguished with the down-arrow, obviously contains a list of items that are revealed when clicked. Second, a simplified bookmark icon, used throughout our application and nearly universal in its readability, indicates favorites or bookmarks. We think that anyone looking at this control will know that it's a menu that contains favorites. Or, at least, we hope so.

(A note about the design of the bookmark icon itself: we modeled its gray, flat appearance off the "utility buttons" Apple has introduced in applications like iPhoto. But, as a twist, we shaded the icon with glossy white highlights to match Mac OS X 10.2's pop-up menu itself. You won't find this kind of stuff anywhere in the HIG, but darned if it doesn't feel right!)

Before you go whipping up some kind of crazy circular scrollbar, remember: you should rarely, if ever, find yourself rolling your own solution, and if you find yourself inventing your own controls more often than using Apple's, you're either doing something very wrong, or working a new version of Kai's Power Tools.

Good Luck

Here at Panic we've always believed that a truly great application is 50% engineering and 50% design – with any luck, you're now ready to tackle that last 50% and bring every corner of your application up to spec. With a little interface intuition, intense attention to detail, and good old-fashioned Interface Buildin', you can make your application shine as brightly as it deserves to under Mac OS X. It's worth the effort, and believe me – your crazy users will thank you!

References

http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/index.html

By Fritz Anderson, Chicago, Illinois

HTTP POST Queries from Cocoa Applications

Integrating web content with desktop applications
Part 3 in a 3-part series

Introduction

In the first two articles in this series, we saw how easy it was to incorporate high-quality support for web content in Cocoa applications, so long as the request for the content could be fully expressed in the URL. More sophisticated requests, embodied in HTTP **POST** requests, are not directly available from the Cocoa API. Last month, we began exploring how to add such a facility to Cocoa, using the CFNetwork package of Core Foundation to build a formatted **POST** request.

This month, we will put what we've learned into an Objective-C class, that can be initialized with a URL, accept query parameters, present the query, and return the result.

As in the first two articles, our target web site will be *Thomas*, the Library of Congress's database of legislative history, at http://thomas.loc.gov/. We'll build a little application that takes the number of a Congress (for instance the 107th Congress that sat 2001–02) and the name of a member of the House of Representatives, and displays a web page listing all the measures that member sponsored.

Where we've Been

Last month's article already covered the use of Core Foundation's CFNetwork package to assemble and format a **POST** query. We saw how to marshal query parameters in an **NSMutableDictionary**, and how to marry the parameters and the necessary HTTP headers into a query packet using a **CFHTTPMessage**. The example code from last month resulted in an application that displayed the formatted query, ready for dispatch to a server.

Figure 1. Last month's BuildPOSTQuery application.

CFNetwork also provides a powerful facility for dispatching the query once it's built, and receiving the results. This month, we'll build an Objective-C class, **FAWebPOSTQuery**, around what we've already learned about building **POST** queries, and extend it so that getting web data from a **POST** can be as routine as the existing **GET** queries built into Cocoa.

Setting the Stage

Figure 2 shows the Interface Builder connections for the human-interface side of our test application. It's a tiny variation on the applications we've been building for the last two months—two fields for parameters, an **NSTextView** for results, an **NSButton** to set the query in motion, and, this time, an **NSProgressIndicator** to demonstrate that our program can do other things while the query is in-process. The Fetch button is wired to our controller object's **doQuery:** method. The controller code—the client of our **FAWebPOSTQuery** class—can be found in **Listing 1**.

Fritz Anderson has been programming and writing about the Macintosh since 1984. He works (and seeks work) as a consultant in Chicago. You can reach him at fritza@manoverboard.org.

W WINDOWS | M MACINTOSH | X MAC OS X

Thinkfree

ThinkfreeOffice

COMPATIBLE WITH MICROSOFT WORD, EXCEL AND POWERPOINT

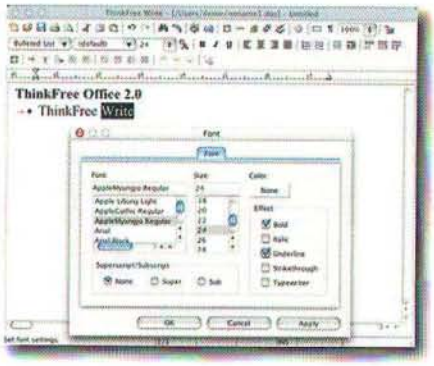
WORDPROCESSOR • SPREADSHEET • PRESENTATION GRAPHICS

The Affordable Office Alternative!

Three High-Performance Applications

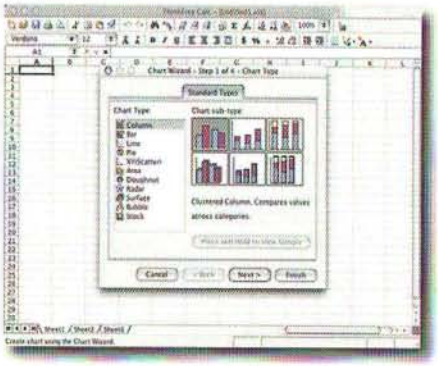
Thinkfree Write

Thinkfree Write is a powerful word processing application that enables you to create rich, professional quality documents and Web pages. You can insert tables, images, and clipart, or even apply custom layouts to your document...then effortlessly proofread your work with the easy-to-use spelling and auto-correction features.



Thinkfree Calc

Thinkfree Calc is a full-featured, easy-to-use spreadsheet application that can easily tackle the most complex analytical tasks with over 40 charts and 300 function capabilities. Thinkfree Calc opens, edits, and saves directly into the Microsoft Excel (.xls) format, so users can seamlessly share documents and collaborate with Microsoft Office users.



Thinkfree Show

Thinkfree Show enables you to create high-impact presentations including animation effects, drawings, images, clipart, and other graphic features. Thinkfree Show opens, edits and saves directly into the Microsoft PowerPoint (.ppt) format.

CyberdrivePlus

A free, one-year subscription to CyberdrivePlus is also included. CyberdrivePlus provides you with secure, Internet file storage and free online software upgrades!

ONLY $49.95

Macworld

"Thinkfree is a best-of-breed program that will exceed your expectations."
— *Jeffery Battersby*

MacAddict RATED GREAT

"Thinkfree Office is the next best thing and then some."
— *Deborah Shadovitz*

macHOME

"Thinkfree Office is an impressive attempt to crack the seemingly impenetrable productivity market."
— *Chris Ward*

amazon.com
Apple Store
Apple Specialist
MacMall
Small Dog Electronics
Fry's ELECTRONICS
ZONES

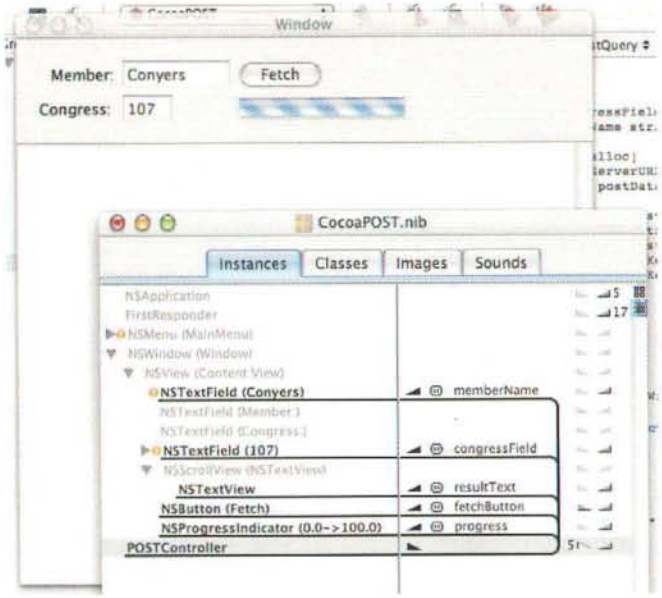


Figure 2. The GET window modified for the POST example

The process of making a **POST** query can be divided into three parts: framing the query, posting the formatted query, and collecting the results. True to our plan, we find that **doQuery:** consists of only two method calls—**frameQuery** and **postQuery**.

Framing the query

The **frameQuery** method starts by harvesting the query parameters, and then creates the **FAWebPostQuery** object. The object needs a URL for initialization, because an **FAWebPostQuery** makes no sense without one, and optionally an **NSDictionary** of keys and values for the body of the query. As we did last month, we marshal the parameters of the query in the query object. Parameters that never vary can be supplied through a constant **NSDictionary**, while parameters that could differ with each query are set with the method **setPostString:forKey:**. Because **POST** queries are key-value lists, **NSDictionary** provides a natural analogy for specifying the body of the query. Using **NSDictionary** for initialization, and a method of the form **set~:forKey:** for management, was therefore an obvious choice in API design.

Sending the Query

POSTController's **postQuery** method is even simpler: It sends the message **post** to the **FAWebPostQuery**. That's it. The rest of the method sets an **NSTimer** to call our **pendingTimer:** method 20 times a second so we can run the **NSProgressIndicator**, and deactivates the Fetch button so we don't have to deal with more than one query at a time.

Receiving the Result

Now all the **POSTController** has to do is let the results roll in. **FAWebPOSTQuery** relies on an informal protocol, that its delegate must implement the method **webPostQuery:completedWithResult:**. When the query has finished—successfully or not—**FAWebPOSTQuery** returns the results and any result code to that method.

In the mean time, **POSTController**'s timer method, and other methods on the UI thread, have been free to update the display and handle user events. So far as the user, and the code that uses **FAWebPOSTQuery** is concerned, the query takes place completely in the background.

Behind the Scenes

That's what happens so far as our UI testbed—**FAWebPostQuery**'s client—is concerned. How is it done?

Last month we went over the issues in using **CFHTTPMessage** to format the POST query. Readers of that article should find the **init...** and **setPostString:forKey:** methods familiar, as well as the initial part of the post method, in which the **CFHTTPMessage** is finished off and readied for sending.

We could have the **CFHTTPMessage** turn over its serialized bytes, send them directly, and handle the rest of the transaction ourselves, but there is a much neater mechanism available, an elaboration of **CFStream**, the Core Foundation implementation of the stream-of-bytes data type.

A **CFReadStream** or **CFWriteStream**, alone, is a routine sort of abstract data type: It allows you to do sequential buffered reads or writes on files, sockets, or memory using calls similar to the POSIX **read(2)** and **write(2)** functions. In this case, the **CFReadStream** we will be using will serve as a read descriptor returning the bytes of the body of the query response, but that is only one of four roles the **CFReadStream** will be playing.

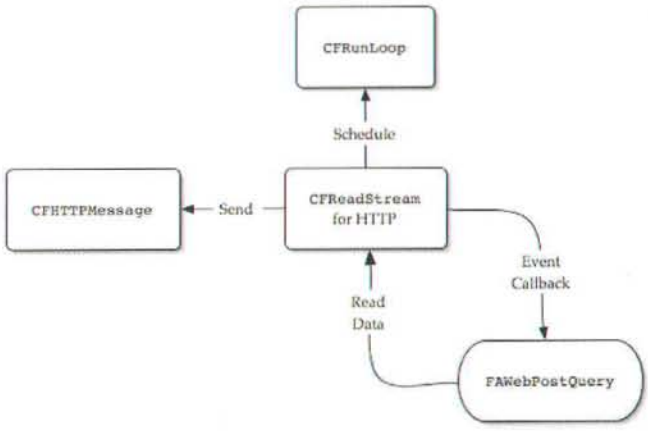


Figure 3. The roles of the CFReadStream

The **CFReadStream** will

- Send the query message to the server.

The power of UNIX brought to the Mac

mac:ODBC
Open database connectivity for the Macintosh

Advanced SQL Editor
Powerful multi-platform and multi-DB SQL script editor

SQL Project
Multi-platform SQL development environment

Team Diagram
Dynamic diagramming and charting tool with CVS and database hooks

Data Architect
Multi-platform suite of database modelling and design tools: includes all the above software components

More Mac OS X products from theKompany.com coming soon

Take advantage of the power of UNIX with theKompany.com's suite of database software. Now also for Mac OS X.

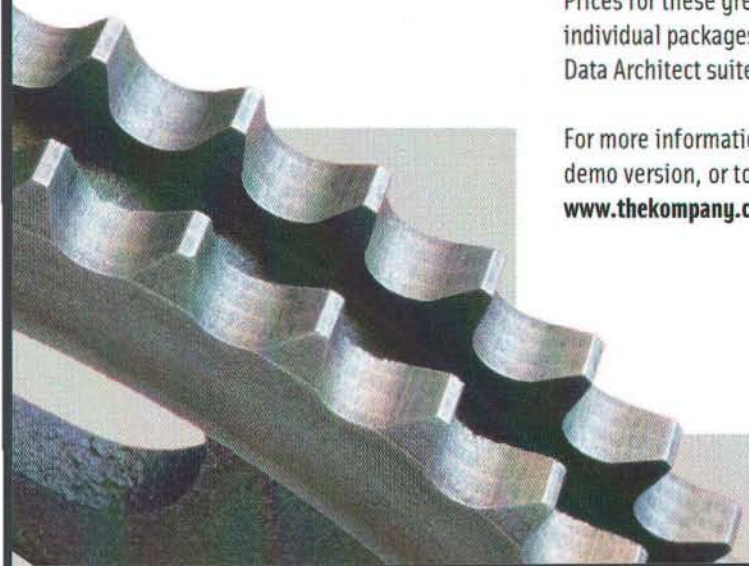
TheKompany.com, a UNIX software and services company, now offers its powerful UNIX database connectivity tools for the Mac OS X computing platform.

With theKompany.com's suite of database tools, you can do just about anything with your SQL databases: connect to SQL databases with mac:ODBC, create and edit SQL scripts with the Advanced SQL Editor, create diagrams from CVS trees and databases with Team Diagram, develop powerful SQL-based projects with SQL Project and more.

Each of these tools is available separately, or as part of Data Architect – theKompany.com's premiere database modelling tool. All are offered with a unique open source license, so you can tweak the program code to your liking.

Prices for these great products start at just **$9.95** for individual packages and top out at **$49.95** for the complete Data Architect suite.

For more information about pricing and availability, to try a demo version, or to place your order online, visit us at **www.thekompany.com**.



theKompany.com

- Make callbacks to **FAWebPostQuery** code when events occur.
- Be a client of the application's run loop, to get a share of processor time to do its work.
- Yield bytes of the response to the query in response to a read call.

Each of these roles has to be initialized and (eventually) torn down.

Role 1: Sender of the query

The process starts with creating a **CFReadStream** for this transaction:

```
replyStream = CFReadStreamCreateForHTTPRequest(
                        kCFAllocatorDefault, message);
CFRelease(message);
message = NULL;
```

Because the **CFReadStream** will be responsible for sending the **POST** query, the first thing we do is to associate the query message with the stream. At this point, **FAWebPostQuery** has no further business with the query message, so it calls **CFRelease** to release its hold on it. This doesn't deallocate the message—the stream still retains a reference to it, and will release its reference later. The message will actually be sent when we call **CFReadStreamOpen()**.

Role 2: Sender of event callbacks

Next, we initialize the stream's role as a sender of event messages to the **FAWebPostQuery**. The **FAWebPostQuery** will want to know when data arrives in response to the query, and when the query transaction has finished—successfully or not.

```
BOOL enqueued = CFReadStreamSetClient(replyStream,
                  kCFStreamEventHasBytesAvailable |
                    kCFStreamEventErrorOccurred |
                    kCFStreamEventEndEncountered,
                  MyReadCallback,
                  &cfContext);
```

CFReadStreamSetClient tells our reply stream what events we are interested in; that we want our function **MyReadCallback** to be called when they happen; and that we want certain context information passed to the callback function.

If you've done much programming with APIs that make callbacks—for instance, the **NSTimer** and **NSNotification** mechanisms in Cocoa—you're familiar with the custom of providing a "user info" pointer in the setup of the callback. It's a way to pass a pointer to an object or other helpful context information into your callback handler. The last parameter of **CFReadStreamSetClient** serves the same purpose, but instead of a simple generic pointer, this parameter must be a pointer to a **CFStreamClientContext** structure.

The reasoning behind this choice was this: The user info that you might want to pass through to a **CFStream** callback might be an ordinary pointer; it might be a reference-counted pointer to a Core Foundation object; or it might be a Cocoa object, which is also reference-counted, but by a different mechanism. The designers of the API decided that the **CFStream** should have a way to retain the user-info object if that is possible. (If you are done with the object, and the CFStream can retain and release it, you can release it immediately and not have the headache of guessing when it will be safe to release it later.) Therefore, you have to wrap the user-info pointer in a structure that includes pointers to functions that retain, release, and provide a **CFString** description of, the user-info data.

(Other Core Foundation APIs that define context structures allow you to pass **NULL** for the retain, release, and description function pointers if you do not want to define these operations. It is fair to assume the same rule applies to **CFStreamClientContext**, but at the time I write this, this part of the CFNetwork API had not yet been fully documented.)

FAWebPostQuery passes itself as the user-info object, and therefore provides C wrappers to its inherited retain, release, and description methods. **Listing 2** provides the whole story.

Role 3: Run loop client

Now we are ready to set the **CFReadStream** for its third role, as a client of the application run loop.

```
CFReadStreamScheduleWithRunLoop(replyStream,
                CFRunLoopGetCurrent(),
                kCFRunLoopCommonModes);
```

Veterans of Mac OS 9 and earlier are familiar with the *event loop*, the heart of a Macintosh program in the old operating system. At the head of the event loop is a call to **WaitNextEvent()**, which returns whenever user input or some other event occurs that the application must process; the rest of the loop is devoted to identifying what part of the application should handle the event, and dispatching control to that handler.

Every thread under Core Foundation and Cocoa has an event loop of its own. At base, it's a **CFRunLoop**, which is *not* toll-free bridged to the Cocoa **NSRunLoop**—you can get the underlying **CFRunLoop** from an **NSRunLoop** with the method **getCFRunLoop**. As with its Mac OS 9 cousin, it waits for events and dispatches them. Unlike the Mac OS 9 event loop, the details of the loop are hidden; the task of calling handlers is done automatically; and the gamut of "events" that can be handled—timer events, driver events, UI events—is practically unlimited.

A **CFRunLoop** waits, without consuming CPU time, until an event occurs that one of its registered clients can handle. Registered clients can include **CFRunLoopTimer**s (or their toll-free equivalents, **NSTimer**s), whose events reflect the expiration of their timers; or I/O objects like our **CFReadStream**, whose events include the arrival of data, end-of-data, or a fatal error. When the event occurs, the **CFRunLoop** calls the client's handler to respond to the event. In our case, **CFReadStream** will, in turn, call our callback function **MyReadCallback**. When the handler is done, control returns to the run loop, which returns to its sleep, waiting for the next event.

In this case, we register the reply stream with the current run loop. By specifying **kCFRunLoopCommonModes**, we ask that the stream's events be handled unless the event loop is put into a mode that explicitly restricts the handling of events. This is the usual way to register a run loop client.

With the call to **CFReadStreamOpen()**, the query is under way, and the response arrives over the next few seconds. Against the possibility it doesn't arrive at all, we set a timeout timer:

```
timeoutTimer = [NSTimer
            scheduledTimerWithTimeInterval: sPostTimeout
            target: self
            selector: @selector(messageTimedOut:)
            userInfo: nil
            repeats: NO];
[timeoutTimer retain];
```

And, whenever the Thomas server gives evidence that it is alive—by sending us data—we restart the timeout clock with:

```
[timeoutTimer setFireDate:
    [NSDate dateWithTimeIntervalSinceNow:
            sPostTimeout]];
```

Role 4: Read bytes from the stream

The remaining business of the **FAWebPostQuery** object is to handle the events that come from the reply **CFReadStream**, all of which come to the callback function **MyReadCallback**. As you'd expect of an event handler, this function is built around a **switch** statement keyed on the type of event that arrived. Because we passed the **FAWebPostQuery** object as the context when we registered the callback, we get it back as a **void*** in the third parameter of the function.

When the event is **kCFStreamEventHasBytesAvailable**, the read stream finally fulfills the one task for which it is named:

```
UInt8    buffer[READ_SIZE];
CFIndex  bytesRead = CFReadStreamRead(stream,
                buffer,
                READ_SIZE-1);
//leave 1 byte for a trailing null.

if (bytesRead > 0) {
  //Convert what was read to a C-string
  buffer[bytesRead] = 0;
  //Append it to the reply string
  [object appendContentCString: buffer];
}
```

Just as we would with a POSIX **read()** call, we pass the stream pointer, a buffer address, and the available length of the buffer to **CFReadStreamRead**. The number of bytes actually read is returned. Then the newly-arrived string can be appended to our results string.

Cleaning up

I decided to handle the other two events, **kCFStreamEventErrorOccurred** and

kCFStreamEventEndEncountered in the same way. Either way, the query is over: All errors in **CFReadStream** handling are irrecoverable. In both cases, there are three tasks to complete: Harvest the last information the **CFReadStream** can yield; tear down the **CFReadStream**; and inform the client of the **FAWebPostQuery** that the query has finished.

We've noticed before that HTTP messages fall into two parts, header and body. The bytes returned to **CFReadStreamRead** while the response was read were, in fact, *only* the body of the response. The header of the response is kept as an attribute of the **CFReadStream**, and it is possible to get the response status code thus:

```
CFHTTPMessageRef  reply =
  (CFHTTPMessageRef) CFReadStreamCopyProperty(
    replyStream,
    kCFStreamPropertyHTTPResponseHeader);

// Pull the status code from the headers
if (reply) {
  statusCode =
    CFHTTPMessageGetResponseStatusCode(reply);
  CFRelease(reply);
}
```

Now that we're done with the reply stream, we can release it from its various roles, and deallocate it. The first step is **CFReadStreamClose()**, which stops all further actions by the reply stream. Next, we tear down its role as a provider of callbacks to the **FAWebPostQuery** object (and release any retains it might have done on the query object) by calling **CFReadStreamSetClient()** with a **NULL** client value. Finally, we remove the reply stream from its role as a client to the run loop with **CFReadStreamUnscheduleFromRunLoop()**. The reply stream can now be released with **CFRelease()**.

Conclusion

Those of us who met Cocoa for the first time in Mac OS X were amazed at the rich toolbox Apple laid at the feet of developers. The riches are still coming in the form of Core Foundation APIs like **CFNetwork**. Core Foundation can be daunting at first, but it's so well-designed that every Cocoa programmer should consider adding it to his repertoire.

Listing 1a: POSTController.h

```
//
// POSTController.h
// CocoaPOST
//

#import <Cocoa/Cocoa.h>

@class FAWebPostQuery;
```

class POSTController

```
@interface POSTController : NSObject {
  IBOutlet NSTextField *          memberName;
  IBOutlet NSTextField *          congressField;
  IBOutlet NSProgressIndicator *  progress;
  IBOutlet NSButton *             fetchButton;
  IBOutlet NSTextView *           resultText;

  FAWebPostQuery *                  theQuery;
  NSTimer *                         busyTimer;
}

- (IBAction) doQuery: (id) sender;

- (void) frameQuery;
- (void) postQuery;
- (void) webPostQuery: (FAWebPostQuery *) query
        completedWithResult: (int) code;
- (void) pendingTimer: (NSTimer *) aTimer;

@end
```

Listing 1b: PostController.m

```
//
// POSTController.m
// CocoaPOST
//

#import "POSTController.h"
#import "FAWebPostQuery.h"
```

sConstantDictionary, sThomasURL

Once initialized, these will be a dictionary and URL that constitute the parts of a member-name query that don't vary. The initialize class method will fill them in. They will be used in the frameQuery method to initialize new FAWebPostQuery objects.

```
static NSDictionary * sConstantDictionary = nil;
static NSURL *        sThomasURL = nil;
```

class POSTController

```
@implementation POSTController
```

initialize

"Initialize" class methods are called automatically by the Objective-C runtime before any of the class's methods are executed. In this method, the static NSURL and NSDictionary that constitute the invariant parts of a Thomas query by member name are initialized.

```
+ (void) initialize
{
  if (! sConstantDictionary) {
    sConstantDictionary = [[NSDictionary alloc]
      initWithObjectsAndKeys:
      @"bimp", @"TYPE1",
      @"SPON", @"Sponfld",
      nil];

    sThomasURL = [[NSURL alloc]
      initWithString:
        @"http://thomas.loc.gov/cgi-bin/bdquery"];
  }
}
```

doQuery

This is the action method for the window's "Fetch" button. It initializes a new FAWebPostQuery from the information entered in the window's fields, and then sends the query to the Thomas server.

```
- (IBAction) doQuery: (id) sender
{
  [self frameQuery];  // Initialize the query
  [self postQuery]; // Execute the query
}
```

stopTimer

A convenience method that checks to see if a query-pending timer is active, and if so, unschedules it from the run loop.

```
- (void) stopTimer
{
  if (busyTimer) {
    [busyTimer invalidate];
```

```
    busyTimer = nil;
  }
}
```

dealloc

The standard release-of-resources handler for deallocation time.

```
- (void) dealloc
{
  [self stopTimer];
  [theQuery release];
}
```

frameQuery

Harvest the number of the Congress and the name of the Congressman from the respective fields, and initialize an FAWebPostQuery to suit.

```
- (void) frameQuery
{
  int      congress = [congressField intValue];
  NSString * member = [memberName stringValue];

  theQuery = [[FAWebPostQuery alloc]
              initWithServerURL: sThomasURL
                       postData: sConstantDictionary];

  [theQuery setPostString:
        [NSString stringWithFormat: @"d%d", congress]
    forKey: [NSString stringWithFormat: @"Dbd%d",
                                        congress]];

  [theQuery setPostString:
        [NSString stringWithFormat: @"/bss/d%dquery.html",
                                        congress]
    forKey: @"srch"];

  [theQuery setPostString: member forKey: @"HMEMB"];

  [theQuery setDelegate: self];
}
```

postQuery

Tells the FAWebPostQuery we're done with preparation, and that it should send the query. We start an NSTimer to get periodic opportunities to animate our progress indicator while the query is in progress. Finally, we deactivate the "Fetch" button, because I don't want to support multiple or interrupted queries.

```
- (void) postQuery
{
  [theQuery post];
  busyTimer = [NSTimer scheduledTimerWithTimeInterval: 0.05
              target: self
              selector: @selector(pendingTimer:)
              userInfo: nil
              repeats: YES];
  [fetchButton setEnabled: NO];
}
```

webPostQuery:completedWithResult:

The query has completed. If it resulted in an error, inform the user. Otherwise, harvest the HTML in the reply and display it. This is the required method from the FAWebPostDelegate informal protocol.

```
- (void) webPostQuery: (FAWebPostQuery *) query
  completedWithResult: (int) code
{
  [self stopTimer];

  if (code == 200) {
    NSString *        result = [theQuery replyContent];
    NSData *          theHTML = [result dataUsingEncoding:
                                 NSASCIIStringEncoding];
    NSAttributedString *  styledText =
        [[NSAttributedString alloc]
            initWithHTML: theHTML documentAttributes: nil];

    [[resultText textStorage]
        setAttributedString: styledText];
  }

  [theQuery release];
  theQuery = nil;
  [fetchButton setEnabled: YES];
}
```

pendingTimer:

The callback for the timer that this POSTController runs while awaiting completion of the query. It is here solely to run the progress bar, as a demonstration that the program is free to do other work while the query is being processed. The FAWebPostQuery object keeps a timer of its own for timeouts on the query.

```
- (void) pendingTimer: (NSTimer *) aTimer
{
  [progress animate: nil];
}

@end
```

Listing 2a: FAWebPostQuery.h

```
//
// FAWebPostQuery.h
// Jefferson
//

#import <Foundation/Foundation.h>

enum {
  FAWebPostIncomplete = -1,
  FAWebPostNotReplied = -2,
  FAWebPostReplyInProgress = -3,
  FAWebPostInvalid = -4,
  FAWebPostTimedOut = -5
};
```

FAWebPostAlreadyPosted

An exception that is thrown if an attempt is made to dispatch an FAWebPostQuery that has already sent its query.

```
extern NSString * const FAWebPostAlreadyPosted;
```

class FAWebPostQuery

```
@interface FAWebPostQuery : NSObject {
  CFHTTPMessageRef        message;
  CFReadStreamRef         replyStream;
  NSMutableDictionary *   postData;
  int                     statusCode;
  CFMutableStringRef      cfReplyContent;
  NSTimer *               timeoutTimer;

  NSObject *              delegate;
  CFStreamClientContext   cfContext;
}

- (id) initWithServerURL: (NSURL *) server;
- (id) initWithServerURL: (NSURL *) server
                postData: (NSDictionary *) initialData;
- (void) setPostString: (NSString *) string
                forKey: (NSString *) key;

- (void) post;
- (void) cancel;

- (int) statusCode;
- (NSString *) replyContent;
- (NSObject *) delegate;
- (void) setDelegate: (NSObject *) aDelegate;

@end
```

FAWebPostDelegate

This is an informal protocol that must be implemented by any object that is passed to the setDelegate: method of an FAWebPostQuery. It declares the signature of the callback message that is sent to the delegate object when the query either completes or fails.

```
@interface NSObject (FAWebPostDelegate)

- (void) webPostQuery: (FAWebPostQuery *) query
  completedWithResult: (int) code;

@end
```

Listing 2b: FAWebPostQuery.m

```
//
// FAWebPostQuery.m
// Jefferson
//

#import "FAWebPostQuery.h"
#import  "httpFlattening.h"

#define  READ_SIZE      1024

NSString * const  FAWebPostAlreadyPosted
          = @"FAWebPostAlreadyPosted";

static CFTimeInterval sPostTimeout = 15.0;

// A template for HTTP stream-client contexts
static CFStreamClientContext sContext = {
  0, nil,
  CFClientRetain,
  CFClientRelease,
  CFClientDescribeCopy
};
```

class FAWebPostQuery

```
@implementation FAWebPostQuery
```

timeoutInterval

A class method that returns the interval, in seconds, after which an HTTP connection is considered to have timed out. When a query is posted, the reply must begin arriving within this time interval, and gaps between batches of data may not last longer. If the timer runs out, the connection is closed and the query fails with the error status FAWebPostTimedOut.

```
+ (CFTimeInterval) timeoutInterval
{ return sPostTimeout; }
```

setTimeoutInterval

Sets the length, in seconds, for all timeout intervals beginning after this class method is called.

```
+ (void) setTimeoutInterval: (CFTimeInterval) newInterval
{
  sPostTimeout = newInterval;
}
```

CFClientRetain

A glue function bridging the Objective-C FAWebPostQuery object to Core Foundation. A pointer to this function goes into the retain field of the client context for the HTTP CFStream that services the reply to the query.

```
void *
CFClientRetain(void * selfPtr)
{
  FAWebPostQuery *  object
        = (FAWebPostQuery *) selfPtr;

  return [object retain];
}
```

CFClientRelease

A glue function bridging the Objective-C FAWebPostQuery object to Core Foundation. A pointer to this function goes into the release field of the client context for the HTTP CFStream that services the reply to the query.

```
void
CFClientRelease(void *selfPtr)
{
  FAWebPostQuery *  object
        = (FAWebPostQuery *) selfPtr;

  [object release];
}
```

CFClientDescribeCopy

A glue function bridging the Objective-C FAWebPostQuery object to Core Foundation. A pointer to this function goes into the copyDescription field of the client context for the HTTP CFStream that services the reply to the query.

```
CFStringRef
CFClientDescribeCopy(void *  selfPtr)
```

REAL Software and MacTech present the REALbasic Showcase to highlight some of the fantastic solutions created by REALbasic users worldwide. The showcase illustrates the wide range of applications that developers using REALbasic can create. Some benefit any Mac user, and others are more specific. All of them are seriously cool!

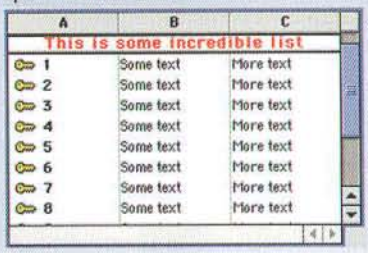
REALbasic is the powerful, easy-to-use tool for creating your own software for Macintosh, Mac OS X, and Windows. It runs natively on Mac OS X as well as earlier versions of the Mac OS. For more information, please visit: **<www.realbasic.com>**.

The Made with REALbasic program is a cooperative effort between REALbasic users and REAL Software, Inc. to promote the products created using REALbasic and the people who create them. For more information about the Made with REALbasic program, please visit: **<www.realbasic.com/realbasic/mwrb/Partners/MwRbProgram.html>**.

Extend REALbasic with Advanced Plugins

Spreadsheet Controls:

We provide a wide range of spreadsheet controls for REALbasic, including multistyled and custom rendering spreadsheet controls.



- More than 32k of rows.
- Classic, OS X and Win32.
- Accelerated for maximum speed.
- Images in cells.

Cryptography:

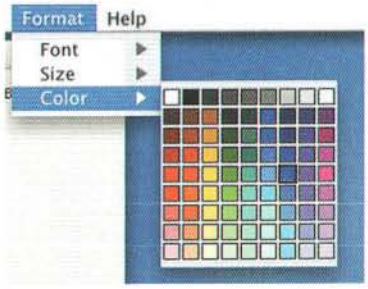
We provide data encryption, encoding, compression and hashing for REALbasic.

Supported Algorithms:

- Encryption:
  - e-CryptIt
  - BlowFish (448 bit)
  - AES (Rijndael) (256 bit)
- Encoding:
  - e-CryptIt Flexible
  - Base 64
  - BinHex
  - MacBinary III
  - AppleSingle / Double
  - UUCoding
- Compression:
  - Zip on Strings and streams (.gz)
- Hashing and Checksums:
  - CRC32 / Adler32
  - MD5 / HMAC_MD5
  - SHA / SHA1 / HMAC_SHA1

Other Plugins:

We have many other plugins for REALbasic, including plugins to do advanced MacOS Toolbox tasks and more custom Controls.



Speed up developement and make more advanced applications by using plugins ! Get free demos at www.einhugur.com

Einhugur Software
sales@einhugur.com
www.einhugur.com

piPop

Pop-up Hierarchical File Navigation and Launcher

TelnetLauncher

Bookmark and Launch your Telnet and SSH sessions

SimpleKeys

Set your Function Keys to type stuff for you!

piDog Software

http://www.pidog.com/

TitleTrack™ JUKEBOX

MAC MUSIC MANAGEMENT FOR SONY® 5 TO 400 DISC CD CHANGERS

Control up to 12 changers (4,800 music CDs)
Control Any Brand Stereo Receiver
Play MP3 and other Sound Files
Plus much, much more!!!

www.titletrack.com **info@titletrack.com**

```
{
  FAWebPostQuery *  object
        = (FAWebPostQuery *) selfPtr;

  return (CFStringRef) [[object description] retain];
}
```

getResultCode

An internal-use method, called when the reply stream has indicated that it has either finished or experienced a fatal error. Retrieves the http header from the reply stream, if possible, and the http result code from the header. Sets the FAWebPostQuery's status code to the result code.

```
- (void) getResultCode
{
  if (replyStream) {
    // Get the reply headers
    CFHTTPMessageRef  reply =
      (CFHTTPMessageRef) CFReadStreamCopyProperty(
        replyStream,
        kCFStreamPropertyHTTPResponseHeader);

    // Pull the status code from the headers
    if (reply) {
      statusCode =
        CFHTTPMessageGetResponseStatusCode(reply);
      CFRelease(reply);
    }
  }
}
```

closeOutMessaging

An internal-use method, called when the CFReadStream that manages the queery reply is no longer needed—either because the whole reply has been received or because the request has failed. This method tears down the stream, the original POST query, and the timeout timer.

```
- (void) closeOutMessaging
{
  if (replyStream) {
    // Close the read stream.
    CFReadStreamClose(replyStream);
    // Deregister the callback client (learned this from WWDC session 805)
    CFReadStreamSetClient(replyStream, 0, NULL, NULL);
    // Take the stream out of the run loop
    CFReadStreamUnscheduleFromRunLoop(
          replyStream,
          CFRunLoopGetCurrent(),
          kCFRunLoopCommonModes);
    // Deallocate the stream pointer
    CFRelease(replyStream);
    // Throw the spent pointer away
    replyStream = NULL;
  }

  if (timeoutTimer) {
    [timeoutTimer invalidate];
    [timeoutTimer release];
    timeoutTimer = nil;
  }
}
```

informDelegateOfCompletion

This method gets called when the query has completed, successfully or not, after the network streams have been torn down. If this object's client has set a delegate, inform the delegate of completion through the method webPostQuery:completedWithResult:.

```
- (void) informDelegateOfCompletion
{
  if (delegate) {
    NSAssert(
      [delegate respondsToSelector:
        @selector(webPostQuery:completedWithResult:)],
      @"A web-POST query delegate must implement "
      @"webPostQuery:completedWithResult:");
    [delegate webPostQuery: self
       completedWithResult: statusCode];
  }
}
```

appendContentCString:

An internal method called by MyReadCallback. It appends the C string it is passed to the CFMutableString that keeps the body of the reply to the query. Passing this message sets this object's status to in-progress, and restarts the timeout timer.

```
- (void) appendContentCString: (char *) cString
{
  CFStringAppendCString(cfReplyContent,
                        cString,
                        kCFStringEncodingASCII);
   statusCode = FAWebPostReplyInProgress;
  // Refresh the timeout timer.
  [timeoutTimer setFireDate:
      [NSDate dateWithTimeIntervalSinceNow:
              sPostTimeout]];
}
```

MyReadCallback

This is the registered event callback for the CFReadStream that manages sending the query and receiving the reply. If data has arrived in the reply, the data is taken from the stream and accumulated. If the transaction ends because of error or success, a final result code is set, the CFReadStream is torn down, and the registered client, if any is informed.

```
void
MyReadCallback(CFReadStreamRef stream,
                      CFStreamEventType type,
                      void *       userData)
{
  FAWebPostQuery *  object =
              (FAWebPostQuery *) userData;

  switch (type) {
  case kCFStreamEventHasBytesAvailable: {
    UInt8    buffer[READ_SIZE];
    CFIndex  bytesRead = CFReadStreamRead(stream,
                                buffer, READ_SIZE-1);
    // leave 1 byte for a trailing null.
    if (bytesRead > 0) {
      // Convert what was read to a C-string
      buffer[bytesRead] = 0;
      // Append it to the reply string
      [object appendContentCString: buffer];
    }
  }
    break;
  case kCFStreamEventErrorOccurred:
  case kCFStreamEventEndEncountered:
    [object getResultCode];
    [object closeOutMessaging];
    [object informDelegateOfCompletion];
    break;
  default:
    break;
  }
}
```

messageTimedOut:

The callback for the internal timeout timer. This method gets called only in the exceptional case of the remote server not responding within the specified time. It's a fatal error, and causes the connection to be torn down and the delegate (if any) notified.

```
- (void) messageTimedOut: (NSTimer *) theTimer
{
  statusCode = FAWebPostTimedOut;
  [self closeOutMessaging];
  [self informDelegateOfCompletion];
}

- (id) initWithServerURL: (NSURL *) server
{
  return [self initWithServerURL: server postData: nil];
}

- (id) initWithServerURL: (NSURL *) server
                postData: (NSDictionary *) initialData
{
  replyStream = NULL;
```

```
    cfReplyContent = CFStringCreateMutable(
                                kCFAllocatorDefault, 0);
    statusCode = FAWebPostIncomplete;
    cfContext = sContext;
    cfContext.info = self;
    timeoutTimer = nil;
    if (initialData)
      postData = [[NSMutableDictionary alloc]
                        initWithDictionary: initialData];
    else
      postData = [[NSMutableDictionary alloc]
                                    initWithCapacity: 8];

    if (!postData) {
      [self release];
      return nil;
      }

    // Set up the POST message and its headers
    message = CFHTTPMessageCreateRequest(
                              kCFAllocatorDefault,
                              CFSTR("POST"),
                              (CFURLRef) server,
                              kCFHTTPVersion1_1);
    if (!message) {
      [self release];
      return nil;
      }

    CFHTTPMessageSetHeaderFieldValue(message,
                    CFSTR("User-Agent"),
                    CFSTR("Generic/1.0 (Mac_PowerPC)"));
    CFHTTPMessageSetHeaderFieldValue(message,
            CFSTR("Content-Type"),
            CFSTR("application/x-www-form-urlencoded"));
    CFHTTPMessageSetHeaderFieldValue(message,
            CFSTR("Host"), (CFStringRef) [server host]);
    CFHTTPMessageSetHeaderFieldValue(message,
                    CFSTR("Accept"), CFSTR("text/html"));

    return self;
}

- (void) setPostString: (NSString *) string
                forKey: (NSString *) key
{
    [postData setObject: string forKey: key];
}

- (void) dealloc
{
    if (message) {
      CFRelease(message);
      message = NULL;
    }

    [postData release];
    if (cfReplyContent) {
      CFRelease(cfReplyContent);
      cfReplyContent = NULL;
    }

    if (timeoutTimer) {
      [timeoutTimer invalidate];
      [timeoutTimer dealloc];
      timeoutTimer = nil;
    }
}

- (void) post
{
    if (statusCode != FAWebPostIncomplete)
      [NSException raise: FAWebPostAlreadyPosted
          format: @"This query has already been posted "
                  @"and either answered or refused."];

    statusCode = FAWebPostNotReplied;

    // String-out the postData dictionary
    NSString * postString = [postData webFormEncoded];
    NSData * postStringData = [postString
```

```
        dataUsingEncoding: kCFStringEncodingASCII
        allowLossyConversion: YES];

// Put the post data in the body of the query
CFHTTPMessageSetBody(message,
            (CFDataRef) postStringData);
// Now that we know how long the query body is, put the length in the header
CFHTTPMessageSetHeaderFieldValue(message,
                            CFSTR("Content-Length"),
    (CFStringRef) [NSString stringWithFormat: @"%d",
                            [postStringData length]]);

// Initialize the CFReadStream that will make the request and manage the reply
replyStream = CFReadStreamCreateForHTTPRequest(
                    kCFAllocatorDefault, message);
// I have no further business with message
CFRelease(message);
message = NULL;

// Register the CFReadStream's callback client
BOOL enqueued = CFReadStreamSetClient(replyStream,
            kCFStreamEventHasBytesAvailable |
              kCFStreamEventErrorOccurred |
              kCFStreamEventEndEncountered,
            MyReadCallback,
            &cfContext);

//    Schedule the CFReadStream for service by the current run loop
CFReadStreamScheduleWithRunLoop(replyStream,
              CFRunLoopGetCurrent(),
              kCFRunLoopCommonModes);

// Fire off the request
CFReadStreamOpen(replyStream);
// Watch for timeout
timeoutTimer = [NSTimer
            scheduledTimerWithTimeInterval: sPostTimeout
            target: self
            selector: @selector(messageTimedOut:)
            userInfo: nil
                    repeats: NO];
  [timeoutTimer retain];
}

- (void) cancel
{
  NSAssert(replyStream,
          @"The program should prevent cancelling "
          @"when no query is in progress.");
  [self closeOutMessaging];
  statusCode = FAWebPostInvalid;
}

- (int) statusCode { return statusCode; }
- (NSString *) replyContent {
  return (NSString *) cfReplyContent;
}

- (NSObject *) delegate { return delegate; }
- (void) setDelegate: (NSObject *) aDelegate
{ delegate = aDelegate; }

@end
```

By Vicki Brown

Transferring Files for Over a Decade

One of my oft-used apps

File Transfer with FTP

Mac OS X provides many choices for transferring files, using FTP and other protocols. Command line tools include curl, fetch, and ftp; GUI-based applications include Fetch (no relation to the command line tool), Interarchy, MacSFTP, and RBrowser. I've tried (and occasionally used) most of the aforementioned tools. However, for general use and drag&drop simplicity, my tool of choice has always been Fetch.

Originally created for internal use by Dartmouth College, Fetch has been available as Mac OS shareware for over a decade. Two years ago, Fetch's author purchased the rights from Dartmouth College, making Fetch an independent product, still shareware. Fetch 4.0 is a Carbon application, providing FTP support under both Mac OS 9 and Mac OS X.

Fetch 4.0 includes all of the expected features of an FTP client, presented in a straightforward interface that mimics the Finder's "list view". Using Fetch is as simple as pressing a button (specifically, the "Get..." and "Put..." buttons). Even simpler, just grab a file and drag it into (or out of) the Fetch window. Files can even be transferred between servers by dragging between Fetch windows; there's no need to save a temporary copy on the local disk.

Fetch claims the distinction of being the first Mac OS FTP client to resume downloads; in version 4.0, that functionality is available even after a crash or quit in mid-transfer. In the "set it and forget it" category, Fetch's "Automatic" transfer mode includes sophisticated logic to choose the appropriate file format or transfer mode ("raw/binary" mode or "text" mode). I've found that "automatic" usually guesses correctly.

Fetch provides many useful features beyond simple Get and Put of files. These include mirror transfers (synchronizing a desktop folder with the server in one command), file management capabilities (creation, deletion, and renaming of remote files and directories from the Fetch window), "Get Info..." and "Set Permissions..." commands for remote files and directories, and easily maintainable lists of shortcuts to commonly used servers.

One of my favorite features is the way Fetch handles URLs. If you paste an FTP URL (e.g. ftp://ftp.gnu.org/gnu/Manuals) into the "New Connection" dialog, Fetch automatically does the right thing, stripping off the "ftp://" prefix and pasting the host name and initial directory into the appropriate fields of the dialog.

If you're concerned about security, or need to use FTP from behind a firewall, Fetch offers many options. These include proxy servers, keychain, Kerberos, One-Time Passwords, and the ability to respond to "challenge" systems.

Fetch is free for use by students, educators, and tax-exempt charitable organizations. A single-user license for other users costs $25; multiple-license packages are also available. For more information, see the Fetch web site (www.fetchsoftworks.com).

Save 15%!

Get our top color management tips FREE!
Simply visit
www.peachpit.com/mactech/download

Fix Your Worst Color Problems!

Real World experts Bruce Fraser, Chris Murphy, and Fred Bunting show you how!

Real World Color Management

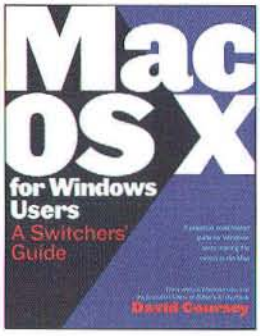
By Bruce Fraser, Fred Bunting, and Chris Murphy

0-201-77340-6 • $49.99

Written by professionals for professionals, this complete reference on color reproduction for print, the Web, and film explains it all. Starting with an in-depth look at color theory, the authors go on to discuss how to get accurate color from today's top design programs. Powerful techniques and tips cover such mission-critical topics as developing a color-management workflow and creating profiles.

Time to Make a Change?

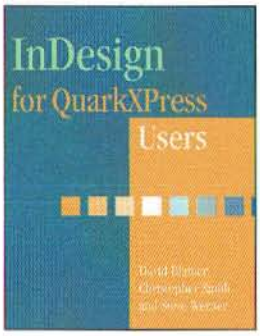
Three new books help you make the switch from PC to Mac, QuarkXPress to InDesign, and film to digital.



Mac OS X for Windows Users: A Switchers' Guide

By David Coursey

0-321-16889-5 • $19.99



InDesign for QuarkXPress Users

By David Blatner, Christopher Smith, and Steve Werner

0-321-15948-9 • $34.99

The Photoshop Book for Digital Photographers

By Scott Kelby

0-7357-1236-0 • $39.99

Save 15% plus free shipping* on these titles and more!

Simply go to www.peachpit.com and enter coupon code EM-43AA-MTMC at checkout.

***Free shipping limited to ground delivery within the United States.**

Peachpit Press

New Riders

ADVERTISER/ PRODUCT LIST

List of Advertisers

List of Products

With rival SCM systems, the only way to quicken the pace is to cut corners - but in the long run you pay the price with missed deadlines, uncertain contents, buggy releases and no way back to previous builds.

With Perforce, the fast way is always the right way. Install it fast, learn it fast, execute operations fast. With other SCM systems, developers face an unpleasant choice: do it the right way or do it the fast way. Perforce's speed and reliability mean fast is right. See how Perforce compares with other leading SCM systems at http://www.perforce.com/perforce/reviews.html

Run at full speed even with hundreds of users and millions of files. At the core of Perforce lies a relational database with well-keyed tables, so simple operations can be accomplished in near-zero time. Larger operations (like labeling a release and branching) are translated into keyed data access, giving Perforce the scalability that big projects require.

Work anywhere. Perforce is efficient over high-latency networks such as WANs, the Internet and even low-speed dial-up connections. Requiring only TCP/IP, Perforce makes use of a well-tuned streaming message protocol for synchronizing client workspace with server repository contents.

Develop and maintain multiple codelines. Perforce Inter-File Branching™ lets you merge new features and apply fixes between codelines. Smart metadata keeps track of your evolving projects even while they develop in parallel.

Truly cross platform. Perforce runs on more than 50 operating systems, including Windows and nearly every UNIX® variation, from Linux® and Mac OS® X to AS/400 and more.

Integrate with leading IDEs and defect trackers: Visual Studio.NET®, Visual C++®, Visual Basic®, JBuilder®, CodeWarrior®, TeamTrack®, Bugzilla™, ControlCenter®, DevTrack® packages, and more.

PERFORCE
SOFTWARE

Fast Software Configuration Management *www.perforce.com*

Download your free 2-user, non-expiring, full-featured copy now from www.perforce.com
Free (and friendly) technical support is on hand to answer any and all evaluation questions.